NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

ENHANCEMENTS FOR THE CAPS PROTOTYPING SYSTEM DESCRIPTION LANGUAGE SYNTAX-DIRECTED EDITOR

by

Scott Robert Grosenheider

March 1996

Thesis Advisors: Shing
Luqi

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	March 1996	Master's Thesis

4. TITLE AND SUBTITLE
Enhancements for the CAPS Prototyping System Description Language Syntax-Directed Editor

5. FUNDING NUMBERS

6. AUTHOR(S)
Grosenheider, Scott Robert

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/ MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/ MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

CAPS (Computer-Aided Prototyping System) is an integrated set of software tools that generate source programs directly from real-time requirements. CAPS users can specify the requirements of prototypes as augmented computational graphs using the graphics/text editor. The problem with the current version of CAPS is that most of the feasibility checks for the prototypes are currently enforced by the translator and the scheduler. Such an approach requires the engineers to go through the "edit, save file, then translate and schedule" cycle in order to find out if the control and timing constraints can be satisfied. The prototyping process can be made much more efficient and user-friendly if these checks are enforced by the CAPS PSDL (Prototype System Description Language) SDE (syntax-directed editor), where users can detect and receive warnings as they enter the design. This thesis focuses on the properties that must exist between processes and their inter-connected data flows in order for a prototype to be correct. It further modifies the PSDL SDE so that parts of the prototype are captured, combined, and manipulated in a way that provides the semantic information needed to determine if these properties have been violated. The new editor has been applied to several prototype examples. The results showed that, by catching errors during the editing phase, the user saves time, is better able to stay focused on the design, and is subsequently more productive.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Synthesizer Generator, Semantic, Syntax, Abstract, Parse, Syntax Directed Editor, Real Time, Timing Constraints	282
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

Approved for public release; distribution is unlimited

ENHANCEMENTS FOR THE CAPS PROTOTYPING SYSTEM DESCRIPTION LANGUAGE SYNTAX-DIRECTED EDITOR

Scott Robert Grosenheider
Captain, United States Marine Corps
B.S., University of Nebraska, 1989

Submitted in partial fulfillment of the
requirements for the degree of

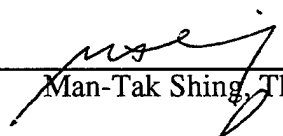
MASTER OF SCIENCE IN COMPUTER SCIENCE

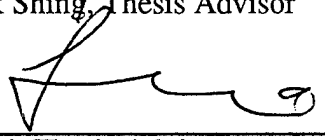
from the

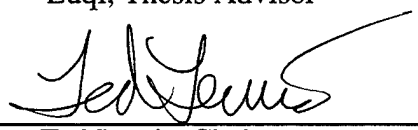
NAVAL POSTGRADUATE SCHOOL

March 1996

Author: 
Scott Robert Grosenheider

Approved by: 
Man-Tak Shing, Thesis Advisor


Luqi, Thesis Advisor


Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

CAPS (Computer-Aided Prototyping System) is an integrated set of software tools that generate source programs directly from real-time requirements. CAPS users can specify the requirements of prototypes as augmented computational graphs using the graphics/text editor. The problem with the current version of CAPS is that most of the feasibility checks for the prototypes are currently enforced by the translator and the scheduler. Such an approach requires the engineers to go through the "edit, save file, then translate and schedule" cycle in order to find out if the control and timing constraints can be satisfied.

The prototyping process can be made much more efficient and user-friendly if these checks are enforced by the CAPS PSDL (Prototype System Description Language) SDE (syntax-directed editor), where users can detect and receive warnings as they enter the design. This thesis focuses on the properties that must exist between processes and their inter-connected data flows in order for a prototype to be correct. It further modifies the PSDL SDE so that parts of the prototype are captured, combined, and manipulated in a way that provides the semantic information needed to determine if these properties have been violated.

The new editor has been applied to several prototype examples. The results showed that, by catching errors during the editing phase, the user saves time, is better able to stay focused on the design, and is subsequently more productive.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

To Dr. Shing, I would like to express my deepest gratitude for being so patient while explaining the required concepts over and over again. Your support, knowledge, and ability to grasp new things never ceases to amaze me.

To my children, Melessa and Kim, who have been known to call me "geek" a time or two in jest, I hope that some day you will be in my shoes. You have given a lot and given up a lot for me and I sincerely appreciate that. I know it is tough, especially for teenagers.

To my beloved wife, Joyce, who has always been there for me, I owe you everything. You mean everything to me and are the reason I have achieved as much as I have. You've given up so much in the past and expected so little in return. Words can't describe my appreciation.

Finally, thank you God for making this all possible.

I. INTRODUCTION

Cheaper, faster, more reliable hardware has helped to make software the dominant factor in today's computing needs. While sky- rocketing demand for software continues, the requirements get much more demanding; especially so for real-time systems with their timing restrictions. Scheduling feasibility is one of the most important and demanding aspects of ensuring that the requirements of a real-time system are met because success means not only being correct, but also on time. Correctness refers to the precision and accuracy which must be adhered to. Being on time refers to the response time required of individual operations. CAPS (Computer-Aided Prototyping System) is an ongoing software engineering project at the Naval Postgraduate School that is targeted primarily at real-time systems. This thesis focuses on some of the issues related to whether or not the design of a real-time prototype is feasible and addresses checking some of these feasibility constraints earlier in the prototype development cycle. It modifies the Syntax-Directed Editor (SDE), which is one of the tools within CAPS, so that errors can be caught while still in the design phase, thereby saving the developer valuable time, enabling more accuracy in the design, and subsequently contributing toward higher productivity.

This chapter describes the current state of software development along with a partial justification for the need of supporting tools and prototyping. Chapter II gives a better description of CAPS, the tools it encompasses, and the steps required to create a prototype. Chapter III discusses the tool that is used to generate the SDE, the Synthesizer Generator. It summarizes the required specifications to creating an SDE. Chapter IV covers creating a short example prototype, some of the constraints that must hold for a real-time prototype to be schedulable, detailed changes that were required of the SDE to implement those constraints, and an example with the new editor. Chapter V summarizes with a conclusion and possible future enhancements.

A. SOFTWARE AFFLICTION

Most people who have had even a small amount of interaction with the development of software have probably heard of the term "software crisis". Many will immediately begin to think of various problems that exist in the field which contribute to this ongoing situation. A

better term for this problem is "chronic affliction" [Ref. 1]. The word "crisis" tends to point to a single point in time where something dramatic happens, where "chronic" means recurring and "affliction" means anything causing pain or stress.

There are many problems in the field of software engineering and most of them are recurring and sometimes very painful. Probably the biggest problem facing software development today is the insatiable appetite for software that our world has amassed; an appetite that is constantly growing. The backlog of demanded systems continues to grow in numbers while the systems themselves are becoming much more complex.

Real-time systems are taking a bigger piece of this pie and among the most difficult to design right because of their restrictions on time. Many are critical in that failure can lead to great costs. History is already full of incidents caused by software failures such as the Bank of New York paying $5 million because of a software problem [Ref. 2] and the Patriot missile that missed a Scud missile because of a timing error and subsequently ended up in 28 Americans killed [Ref. 3]. Quality and user friendliness is expected in degrees rarely achieved to date. All this while many of the current projects are over budget, past due, and not meeting user expectations. Some projects are dropped after extensive amounts of resources have been invested. In fact, as many as 25% of the projects in large MIS organizations are never completed [Ref. 4]. All too often, cost estimates are only guesses because true requirements are simply not known. The bottom line is that the software engineering community must increase productivity dramatically to catch up, let alone keep up.

Probably one of the single most critical areas in which software engineers must get better at is capturing requirements. Most of the time, software products don't satisfy what the user really intended. It may meet the requirements specified by the user, but if it doesn't meet the users needs, its a bust. This is the major cause of unsatisfied customers.

Software engineers must get faster at putting out finished products. Too many times, developers fall behind and this is one of the biggest factors in cost over runs. One well documented fact is that errors get much more expensive in terms of time and money when discovered later in the development cycle. Making matters worse, requirements errors are not likely to be found until implementation. So then, it only stands to reason that the most expensive errors are the ones concerning missed, incorrectly, or inconsistently defined software requirements.

One of the best ways of ensuring that true requirements are captured, thereby enabling the developer to give more accurate time and cost estimates and at the same time maintain a much higher probability that the product will be good (as define by the user), on time, and on budget, is with a technique called prototyping.

B. PROTOTYPING

Prototyping is an excellent tool for determining exactly what is needed by the system to be developed. Its purpose is "to help customers understand and criticize proposed systems and to explore the new possibilities that computer solutions can bring to their problems in a timely and cost effective manner" [Ref. 5]. It is much like the architect who gets some guidance from the client, goes away to draft up preliminary designs, and then comes back to review what has been done. The architect knows that it may be all wrong; better to find out earlier when backtracking is still reasonably cheap than later when it is extremely expensive. Once the designer better understands what the requirements are, more meaningful estimates can be determined along with better cost/benefit driven feasibility studies.

The best situation for prototyping is when there is uncertainty about what is required. This could be uncertainty as to what exactly the customer desires, uncertainty about how to make things happen, or both. In the former case, a client may not know exactly what it is they want, certainly making it difficult for the developer. They may know what is needed but can't describe it. Many times the client speaks in a whole different vocabulary because of her expertise. They may actually or mistakenly have described exactly what is needed and end up with a product that is totally unusable. The English language is very ambiguous at best, leaving lots of room for error.

The second scenario where a software designer is unsure how to make things happen can be a very difficult problem. The designer may not be sure of how to deliver what is wanted in terms of functionality or performance. Understand data and control flow, functional processing, or general understanding of the behavior of the system can be greatly enhanced through prototyping. This situation exists when venturing into unfamiliar terrain and is almost always the case when dealing with real-time systems. The timing constraints imposed can be enormously complex. The client normally can't describe them while the analyst/designer has a tough time trying to determine them [Ref. 3]. Sometimes whole designs must be redone in order

for it to be feasible in terms of scheduling. Many times it is hard to determine if requirements have been met, further justifying the use of prototypes for validation purposes.

While prototyping can be done merely to gather requirements (it is meant to be thrown away after requirements gathering is done), the focus is in terms of a development paradigm. That is the concept of prototyping in a evolutionary approach where the system goes through a continuous cycle of modification/demonstration/evaluation until it is right. An iterative approach that rapidly adjusts the behavior of the prototype based on feedback from the customer and designer. This allows incremental development vice a big bang approach. Most who have worked on a complex project will agree that it is easier to get it right by quickly and correctly doing a little at a time than by trying to do everything at once. The big bang approach gets even more unwieldy when the size dictates multiple designers. With each iteration, the customer and designer come to a more common understanding of what the customer really needs, regardless of preconceived notions by either party. CAPS endorses this evolutionary approach with software tools that support the designer in the task of building and executing prototypes.

One aspect of this process that makes it better than more traditional methods such as the waterfall method is it does not expect to be able to freeze the requirements. Realistically, requirements do not stabilize until the user gets some exposure to the system (especially with real-time systems). With most traditional methods, this means not having stable requirements until after implementation, thereby necessitating significant change during its maintenance phase. Changes that would not have been required had the system been right in the first place. With a prototyping approach, the user gets adequate exposure to the prototype to determine true requirements. And because it is an iterative process, and because the prototype is easier to modify than implemented code, the user will eventually be able to fine-tune tentative requirements into exactly what the system needs are.

This process results in higher quality implementation because fewer initial errors get by. That in turn, lowers maintenance costs, which is the dominant expense in larger systems, because fewer implementation errors means fewer discovered mistakes to correct for during maintenance (many maintenance changes are really errors), not to mention the fights over whose fault and who pays (requirements mistake vs.implementation error). Also, when changes need to be made, the prototype (which is more simple than the production code) can be manipulated/ modified rather than the production code (which is more complex). Then when the required

changes (new requirements) are certain, the new implementation can occur. Much of the implementation may consist of old code but 1) it is a new clean implementation with clear, documented requirements (didn't do patchwork with old implementation) and 2) the changes are easier to achieve as the designer worked with a higher level abstraction.

Because constraints from both the problem domain and the programming domain together drive the requirements of a proposed system, both the user with his knowledge of the problem domain and the designer with her knowledge of the program domain must work together extensively. Prototyping helps create a common ground where they can communicate effectively by demonstrating the behavior of the proposed system. The disagreements that arise quickly uncover areas that need further thought/clarification.

Testing gets more mileage with the iterative approach. Most organizations do the bulk of their testing towards the end of the software development cycle. This means that when funds and time allocated for the development of the system are at the end, the organization is testing to see if the product answers the mail. The question that immediately comes to mind is, "what if it doesn't?". The answer is complex. Fixes can be minor or significant depending on how far off the mark the developer is and how much money, time, and patience the customer has. Most testing authorities will state that because testing is one of the most effective tools in developing correct software, it should be moved further up in the software development cycle. By introducing a prototyping methodology that has validation and verification built into the cycle, this problem can be avoided. Then, when final testing is done, it does not have to be as significant and there are no surprises. This further means that the first implementation will more reflect user needs. That, in tern, means that less modifications must be made later in the maintenance/evolution of the system thereby saving time and money and keeping the system more stable and useful longer.

In a study that looked at thirty-nine significant cases, it was determined that "rapid prototyping is indeed appropriate for large systems, and there seems to be more successful use of evolutionary prototyping than throwaway", and further that "rapid prototyping has had a number of positive effects on both the software product and development process and that it can be used successfully in a variety of situations" [Ref. 6].

Figure 1 below, which was borrowed from the CAPS Tutorial [Ref. 7], helps to show the Prototyping Process. The shaded areas depict the cycle that is executed in order to get the requirements right before implementation.

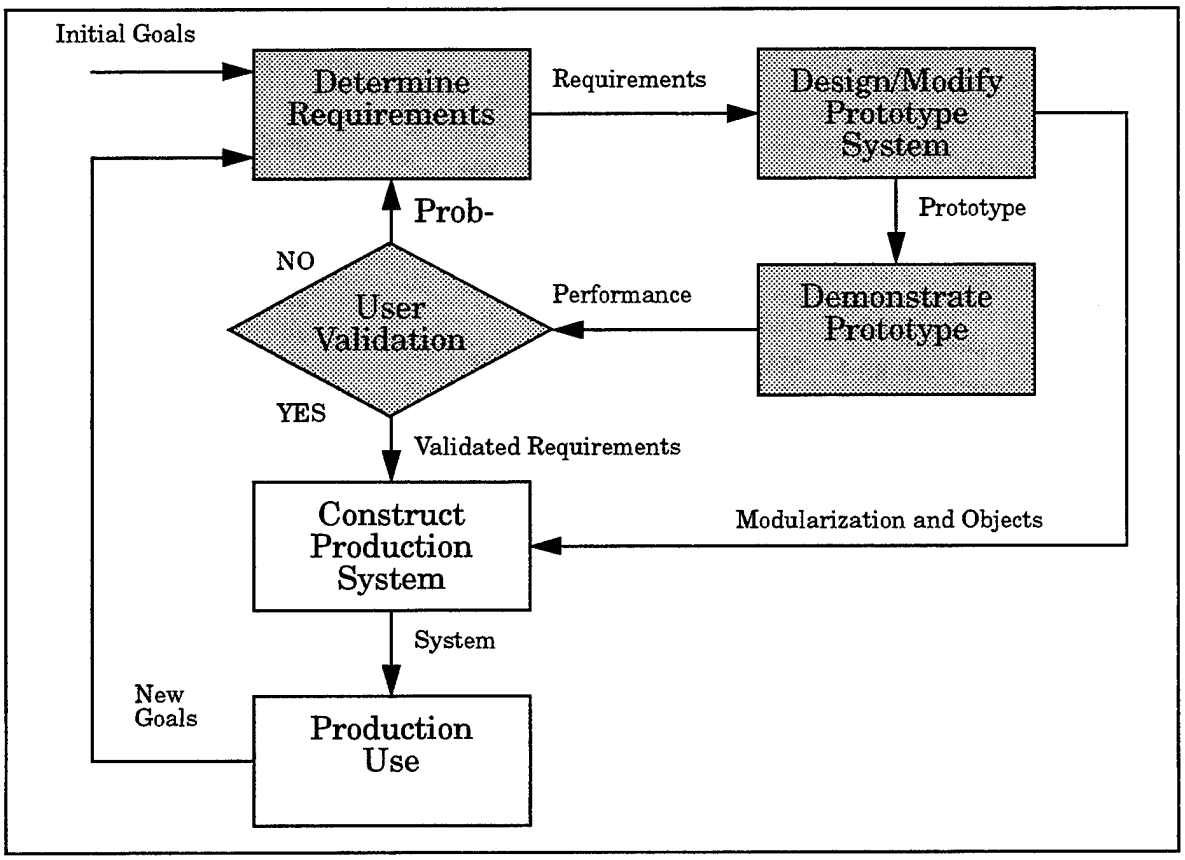


Figure 1. The CAPS Prototyping Process.

II. CAPS

Prototyping can dramatically increase the effectiveness of building and supporting software systems. "A high level language, a systematic prototyping method, and an integrated set of computer-aided prototyping tools are important for realizing the potential benefits of prototyping" [Ref. 8]. The language is PSDL (Prototyping System Description Language) and the integrated set of computer-aided prototyping tools is CAPS (Computer Aided Prototyping System). Useful for any type of application, CAPS is primarily developed for real-time system prototypes and consists of software engineering tools which are linked together by a common interface. These tools assist the designer in constructing and executing the prototype [Ref. 8].

Several sources make reference to hard real-time systems which are those in which deadlines and requirements are guaranteed to be met under the worse-case situation. While some COTS (commercial off the shelf) products are available for real-time support, only CAPS generates code which satisfy the constraints of a hard real-time system [Ref. 3]. The CAPS development environment is displayed in Figure 2 below.

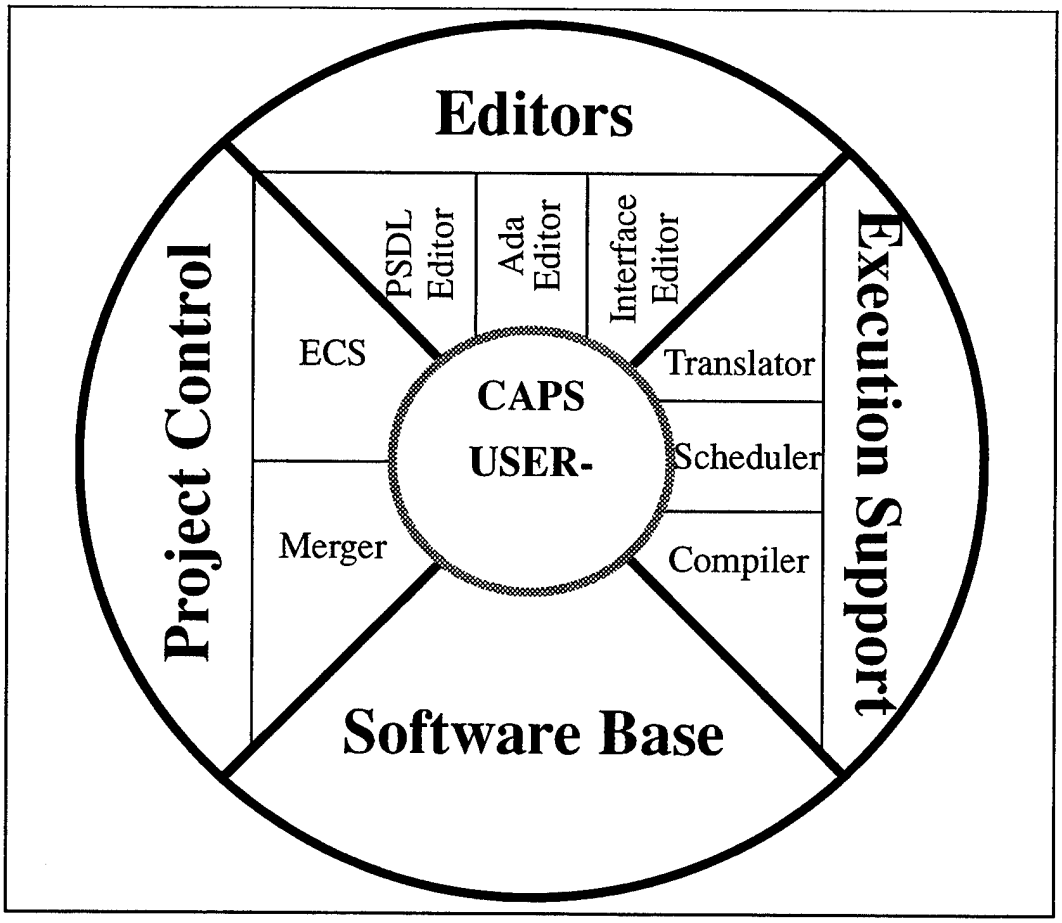


Figure 2. The CAPS Development Environment.

A. PSDL

Most of the tools within CAPS operate on PSDL which is a prototyping language designed to enable a high level description of the system. PSDL is the means by which these tools communicate thereby assisting them in demonstrating the behavior of the prototype. It was designed to assist requirements analysis, feasibility studies, and in designing large embedded systems. Because this thesis involves modifications to the SDE, which is heavily intertwined with PSDL, more detail will be spent on PSDL and the PSDL Editor than other parts of CAPS.

An underlying computational model was chosen to make inter-component communications explicit [Ref. 9]. It is formally represented as an augmented hypergraph, G, where $G=(V,E,T(v),C(v))$ [Ref. 10]

where:

V is a set of vertices (vertices represent operators),

E is a set of edges (edges represent data streams),

$T(v)$ is the set of timing constraints for vertex v, and

$C(v)$ is the set of control constraints for vertex v.

One of the major strengths of CAPS is its support of the prototyping methodology. Prototyping loses much of its appeal when modifications to the prototype are as intensive as modification to production code. Because of this, PSDL was designed to help make the required design modifications, common to the iterative approach, as painless as possible.

Modularity, absolutely necessary to a good design, is supported by operators that explicitly communicate via data streams. A design is represented by a hierarchically structured network of these operators and data streams which are laid out as dataflow diagrams, enhanced with timing and control constraints. Dataflow diagrams convey a great deal of information about the internal structure of a process and yet are simple to read. Extending this diagramming technique with timing and control information made the technique much more powerful while maintaining its simplicity.

PSDL supports reuse by capturing attributes that describe both the interface and the behavior of components. The interface attributes captured include generics, inputs, outputs, states, exceptions, and timing information, while the behavior attributes are keywords, and formal and informal descriptions. These attributes can be used to retrieve reusable components and organize the software base.

Requirements tracing is supported in PSDL by a construct that links the requirements to the part of the prototype that implements it [Ref. 8]. This is important so that as the system evolves, obsolete or changed requirements can quickly be identified in the implementation and modified as necessary. It further helps ensure that the system and its documentation stay up to date.

PSDL further supports the requirements of a real-time system design: Control Constraints, which maintain preconditions on the firing of a module, filter a modules output, and/ or control timers, and Timing Constraints, which implicitly determine when constrained operators will fire, are built into PSDL [Ref. 8]. They are discussed in more detail below. While PSDL was designed with a small set of constructs to make it very powerful, it was kept as simple as possible.

1. Operators

Operators can represent a function or state machine. The act of firing (executing) involves reading one data object from each input data stream and writing zero or one output to each output data stream. The operator represents a function if its output is solely dependent on the inputs to the operator; the same input will always produce the same output. If, on the other hand, its output depends on both inputs and values stored in its memory (internal state), it is a state machine. Operators can affect each other only by explicit data streams and can be further decomposed as design decisions and principles of abstraction dictate.

a. Composite or Atomic

Operators can be classified as composite or atomic. A *composite* operator is one that should be further decomposed. On the flip side, an *atomic* operator is one that should be decomposed no further. When analyzing an operator, the first question is, "Is there a component in the software base that will do this?". If the answer is yes, the newly designated atomic operator is implemented with that component. If, on the other hand, the answer is no, the next question is, "Does it make sense to decompose this operator?". If so, it is dubbed a composite operator and is decomposed. This will result in multiple operators that are each, in turn analyzed with the first question above. Of course, if further decomposition doesn't make sense, the operator is atomic and an implementation must be created.

As with all data flow diagrams, the lowest levels incorporate atomic operators. The goal is to eventually have a significant number of those implementations via components

from the software base, which have a high degree of reliability.

b. Time-Critical or Non Time-Critical

The next classification of an operator is whether it is time-critical or not. Built into the definition of a real-time system is the understanding that there will exist operators that have a constraint on how long they have to complete execution. In PSDL, this is called, its *maximum execution time* (MET) and denotes the maximum amount of CPU time the operator can use for execution. If an operator is assigned a MET, it is obviously *time-critical.* If not, it is simply an operator like the ones we have all seen in normal non real-time systems. Time-critical operators can be broken down into two further categories.

c. Periodic or Sporadic

Again, only time-critical operators (has a MET) are further classified as to whether they are periodic or sporadic. A *periodic* operator has a regular interval in which it must fire and complete its execution. It is assigned a *period* (PER) which denotes the frequency in which the Scheduler makes a processor available for execution. Within that window, the operator must fire and complete execution. It is assigned a finish within (FW) time that denotes how long from the start of the window (PER) before execution must be completed. While the FW starts at the beginning of the PER, the MET doesn't start until the operator fires which can be after the beginning of the PER.

A *sporadic* operator does not necessarily have a regular interval in which it fires and is triggered by the arrival of new data. It is assigned a *maximum response time* (MRT), which is the maximum amount of time between the arrival of new data on the input data stream(s) (which triggers the operator) and the time when the last output is put on the output data streams. In addition, it is assigned a *minimum calling period* (MCP). The MCP is a lower bound on the delay between two subsequent arrivals of triggering data on the input.

More discussion of both periodic and sporadic operators will ensue in chapter five when covering timing and control constraints.

2. Streams

Flows between operators can be data, control, or exception information. A *stream* is a communication link that connects two operators. The originator of the stream is called the *producer* operator while the user of the stream is called the *consumer* operator. A PSDL prototype is schedulable only if the graph is directed and contains no cycles. This is more

commonly referred to as a DAG (directed acyclic graph). When a cycle occurs, it indicates the presence of state information and must be dealt with.

State streams are the means by which state information and hence cycles in the graph are dealt with. The state stream provides a way of declaring and initializing the state. During scheduling, it is actually removed from the graph. Figure 3 shows how streams are depicted in the PSDL Editor of CAPS. Data streams can be broken down into two other classifications. The consumer operator determines what type of stream it is. If the consumer operator has a trigger of "Triggered By All", then the stream is a *data flow stream* (triggering is discussed below). In all other cases, including "Triggered By Some", it is a *sampled stream.*

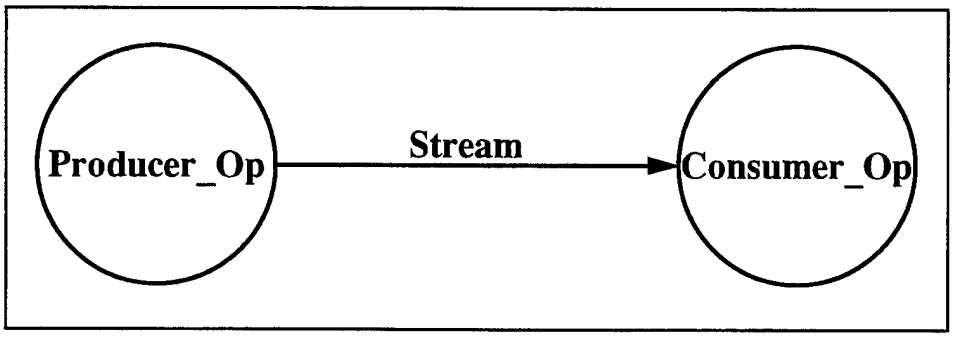


Figure 3. Typical diagram of a stream. Data Streams look as pictured. State Streams have a bold arrowed line connecting the two operators.

A data flow stream guarantees that no data is lost or duplicated [Ref. 10]. It is like a FIFO queue with a length of one. A consumer reads the data from its incoming data stream destructively so that it is no longer there. Underflow occurs if it attempts to read an empty stream. Overflow occurs if the producer operator tries to put new data to the stream before the consumer has read the old data.

Sampled streams do not guarantee against lost or duplicated data. They are like a single memory variable that can be updated or read zero or more times. In addition, reads are not destructive. The only way a data value will go away is if the producer replaces it with another.

3. Timers

Timers are an abstract state machine that act like a stopwatch [Ref. 10]. Used for things such as time-outs or refresh rates, they keep track of the amount of time that elapses between certain events. The four primitive operations are READ, START, STOP, and RESET. These are absolutely essential in a real-time system.

4. Triggers

Any operator can have a trigger and there are two types; *BY ALL* and *BY SOME*. For example, if an operator has "OPERATOR A TRIGGERED BY ALL X, Y", it will fire when new data is on both X and Y data streams. X and Y may or may not be all of the input data streams. This guarantees the output is based on new input and can be used for synchronization purposes. If, on the other hand, it had "OPERATOR A TRIGGERED BY SOME X, Y", it would fire when either data stream X or Y had new data on them.

5. Conditionals

There are two kinds of conditionals in PSDL which control the input and output of an operator. Conditionals can be combined with other constraints including triggers. While these could be implemented within the operator itself, this provides a quick, clear way of controlling the operator, assisting in the goal of making the prototype easier to modify.

a. Conditional Execution

Conditional execution, sometimes called *execution guards,* enforce a pre-condition before allowing the operator to fire. They entail the keyword "IF". The following example shows that the operator will not fire until the condition Brake_On is true: "OPERATOR Suspend_Cruise_Control TRIGGERED IF Brake_On".

b. Conditional Output

Conditional outputs, sometimes called *output guards*, determine whether or not data is written to the output data stream. It does not, however, have any control over the firing of the operator. The condition can depend on operator input/output and timer values.

6. Exceptions

PSDL has a built in abstract data type called exception. It can be used to create exceptions with a given name, detect whether a value is an exception, and determine whether a value is "normal" (a keyword in PSDL). As mentioned previously, they can be transmitted over data streams.

B. EDITORS

There are several editors within CAPS providing a range of functionality. From basic editing of text files to building prototypes. These options/resources fall under the "Edit" pull-down menu.

1. PSDL Editor

A prototype is built with the *PSDL Editor*. It is composed of three parts: the Syntax Directed Editor, the Graph Viewer, and the Graphic Editor. They allow the designer to create the CAPS data flow diagram and PSDL program, assigning all timing and control constraints necessary to ensure the proper design of the prototype and its components.

The *Graphic Editor* is used to build the data flow diagram and to specify some of the timing constraints. This provides a way to show the design structure in a simple diagrammatic way. A great deal of information can be represented in somewhat simple diagrams and modification is quick and simple.

The *Syntax-Directed Editor* (SDE) captures the information already entered into the Graphic Editor. Further, it allows for entering PSDL descriptions that are free of syntax errors by immediately notifying the user when they arise. It is this functionality that is extended by this thesis. The restrictions imposed by the constraints have an impact on whether the prototype is schedulable. Chapter IV covers these restrictions and some of this information has been incorporated in the SDE so that design errors related to timing and control constraints, in addition to syntax errors, are discovered while still in this early phase of design. Chapter III contains an example on creating a prototype.

If, while in SDE, the current position of the prototype pertains to the data flow diagram, the *Graph Viewer* displays that view of the data flow diagram. This helps keep the designer keyed into exactly what part of the prototype s/he is currently in and provides an outstanding overview of what process is currently under review.

2. Text Editor

The text editor can be one of several text editors depending on which one is chosen as the default. CAPS provides a convenient interface to the editor chosen. The user can select which editor is desired by choosing "CAPS Defaults". The possibilities are vi , emacs, and the Verdix Ada Syntax Directed Editor.

3. Interface Editor

Under the "Interface" option, CAPS provides a seamless interface to TAE+, a versatile tool for creating and manipulating dynamic window-based user interfaces. When done, skeleton code is generated, contributing to the goal of generating the prototype quickly and efficiently. TAE+ allows for code to be generated into one file or into multiple files. While generating code under CAPS, the single file option is normally chosen (can be multiple files), and the code is placed in the prototype directory called: <prototype name>.RAW_TAE_INTERFACE.a.

4. Requirements Editor

While the goal is to have a tool that allows mapping from the requirements to the portions of the prototype that implement it, at current the "Requirements" option simply provides a window that lists the files with the extension ".req". From there, any one file can be selected at which time the default editor is utilized to make necessary changes.

5. Change Request Editor

Like the requirements editor, the "Change Request" option brings up a list of files with a specified suffix; this time "cr". Again, the user picks one and the default editor is summoned to edit that file. The hope is for this option to call a sophisticated change request tracking/editing tool.

C. EXECUTION SUPPORT

Rapidly constructing and updating a prototype depends on efficient execution support. These options/resources fall under the "ExecSupport" pull-down menu.

1. Translator

The *Translator* "augments the implementations of the atomic operators and types with code realizing the data streams and activation conditions, resulting in a program in the underlying programming language that can be compiled and executed." [Ref. 10]. Essentially, it generates code that binds the components extracted from the software base or custom built, depending on whether or not they are in the library.

With Ada as the implementation language, it translates PSDL code into Ada wrapper code to realize the control constraints and instantiates Ada tasks for PSDL data streams. It expects a complete PSDL program as input, and creates several packages which make up, in part, the *supervisor module* of the prototype.

2. Scheduler

Because real-time systems have constraints and finite resources that tend to introduce dependencies between the functions of the system that would otherwise be independent, small changes can significantly impact the design [Ref. 3]. And because scheduling is a major factor in correctly designing a real-time system, that step must be automated so that excessive amounts of time are not spent on scheduling with every small change to the design.

The *Scheduler* generates two schedules; a high priority *Static Schedule* and a low priority *Dynamic Schedule*. The former allocates time slots for the time critical operators and if successful (the prototype is schedulable), all operators are guaranteed to meet their required timing constraints. If the prototype is not schedulable, the scheduler gives some diagnostic information that can help the designer see why.

The latter invokes the non constrained operators during execution with the time slots not previously allocated. Translation is required before scheduling which is required before compilation.

3. Compiler

The Compiler option interfaces with the Sun Ada compiler. The prototype must have been successfully translated and scheduled prior to compilation.

D. PROJECT CONTROL

1. Evolution Control System

As the size of a project gets bigger, the number of people working on that project grows. This means more time is spent on communications and less time is spent on analyzing/ designing, etc. Large projects also dictate a longer time until completion which means more turnover of personnel. Both of these problems can be minimized by ensuring that all documentation is stored and managed on-line.

The *Evolution Control System* (ECS)is designed to give automated help to the difficult task of coordinating concurrent efforts of prototype design team(s) and managing the multiple design versions that can be produced. The prototype development data is stored in a design database (DDB) for persistent storage.

2. Merger

The Merger helps to combine the product of two or more independently developed prototype changes thereby facilitating parallel enhancements and applying common changes to multiple versions. It warns of possible conflicts in the merging of two changes and when none exists, will create a PSDL program for the newly created prototype.

E. SOFTWARE BASE

Accessible through the "Databases" pull-down menu, the *software base* is currently designed to provide both "Browse" and "Query" capabilities for accessing a repository of reusable Ada and PSDL components. The user can browse by either types or operators and can query by keyword or PSDL specification.

Standards on how to specify a reusable component are not yet in place. Because of this and the difficult nature of the task, tools for finding and adapting appropriate software components have yet to live up to expectations for strong code reuse. Currently based on parameters, work on this part of CAPS is ongoing to provide better underlying matching capabilities.

III. THE SYNTHESIZER GENERATOR

Attribute Grammars have been used extensively in building compilers with their ability to accomplish translations and specification of static semantic analysis. Attribute Grammars have been used in porting code from legacy systems to newer languages. In general, they have been used in the development of many tools of modern day software engineering in an attempt to provide products that enhance quality and productivity. One of these tools is the Synthesizer Generator which is used to generate the Syntax-Directed Editor (SDE) within the PSDL Editor. A SDE, language-based editor, or smart editor, is an editor tailored to a specific language (in this case, PSDL), utilizing the grammar, structure, and static semantics of that language to assist the user in writing correct programs.

A. BACKGROUND

The *Synthesizer Generator* (SynGen) is a system for developing smart editors which use the knowledge of the language itself to achieve specialization. It automates the implementation of a desired language based editing environment. The knowledge of the language enables the editor, depending on how it is built, to provide feedback to the user concerning whether a program contains syntactic or semantic errors, where they are and recommendations on how to fix them. Inconsistencies can be identified, along with other types of analysis. It can be used for conversions, translations, and transformations. Editors produced can also control how the user proceeds in many various situations. It creates the language-specific editor from a specification of the language's abstract syntax (abstract syntax rules), context-sensitive relationships (Attribute Rules), display format (Unparsing Rules), concrete input syntax (Concrete Rules), and transformation rules (Transformation Rules) and performs analysis, translation, and error reporting with the use of an immediate-computation paradigm [Ref. 13]. The rules identified in parenthesis will be covered individually in order to give a clear representation of the required specifications for a SDE. The immediate-computation paradigm simply means that all attributes are validated with each and every update of the program.

Every object within the program, including the whole program itself, is represented as a consistently attributed derivation tree that goes through many transformations as the program is edited. The rules specified according to the language are used to check these attributes with

every program modification so that the integrity of the tree stays in tact. The language used to do the specifications is called the *Synthesizer Specification Language* (SSL), which is built upon a foundation of attribute grammar, a type definition facility, and the application domain of language-based editors [Ref. 13].

B. USES

The editor created can be a hybrid of many different types of more specialized editors/ tools. Understanding this, those specialized editors and tools should discussed lightly.

Structure editors view a program as a hierarchical composition of individual structures. This means that any component, including the whole program can be broken down into its components. These structures, also called *templates* are predefined formatted patterns representing the constructs in the language; a For Loop for example. Seeing a program this way leads to constructing it by inserting templates into placeholders. For example, if you wanted to insert a FOR LOOP inside an existing IF statement, you would highlight the placeholder inside of the IF statement and insert the FOR LOOP as displayed in Figure 4. If the FOR LOOP template was highlighted and deleted, the old placeholder would return. Obviously all template insertions are controlled and therefore ensure correct syntax.

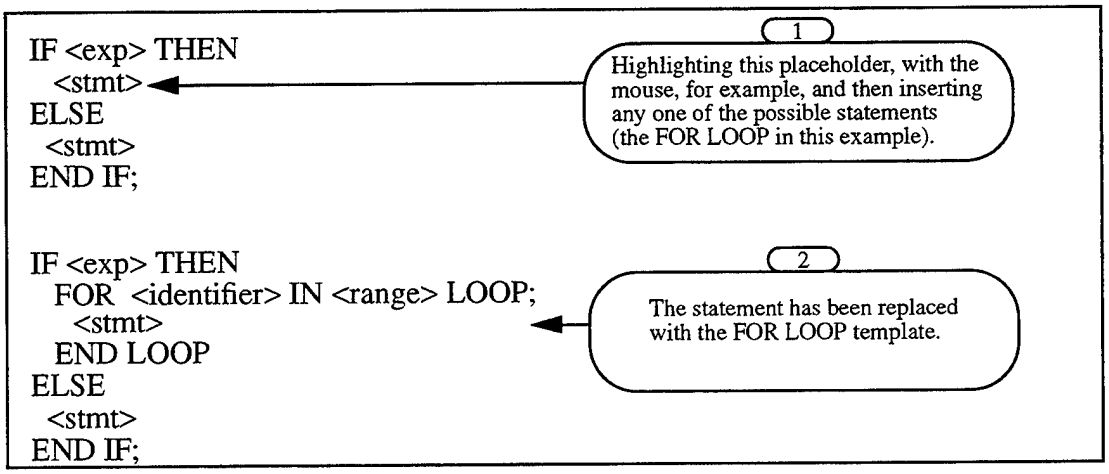


Figure 4. Sample display of using templates.

The typical *text editor* allows for character-oriented and line-oriented textual operations. This is the type of editor that most of us are used to and can be very quick when the user is very familiar with the syntax of the language. SynGen allows for both forms of text operations.

WYSIWYG editors allow for specialized viewing of the information that is stored. They can, as the name implies, show you what you have. They can also display information in a number of ways that can benefit the user; for example, some forms of display are more informative. By allowing for the control of what is displayed, SynGen can achieve variations of WYSIWYG.

Spreadsheet applications are useful in that whenever an update is made in the application, changes are automatically made throughout. In SynGen, this aspect is captured with its immediate-computation paradigm previously mentioned. It enables the editor to complete the analysis, error message generation, and code generation with each incremental change. This is a very important aspect as many types of information can be stored and updated with each modification. Executable code can be output with each change, allowing for immediate feedback on changes made. This becomes a significant productivity enhancement when going through the "modify, run, evaluate" cycle of testing. Correctness and proofs can be supported/ maintained while editing if the rules are spelled out in the SSL.

Incremental code generation was just mentioned above. But what about generating code based on knowledge of another language and a program in the language. This is a hot topic with projects that seek to re-engineer existing systems. There are many legacy systems that have little-to-no documentation. This lack of documentation makes software engineers very reluctant to tackle these systems when it comes to re-writing in newer languages. SynGen can help significantly with this difficult task.

C. BUILDING AN EDITOR

The objective of this chapter is to familiarize the reader with the SynGen and how it works. For a full and in-depth analysis of the SynGen, see references [Ref. 13, 15]. The ideal editor will normally have a combination of all the features discussed above making it a hybrid editor. Building an editor requires that five different specifications be made in SSL. As these specifications are outline, also covered will be terminology of the SynGen. The five specifications, mentioned earlier, are abstract syntax, context-sensitive relationships, display format, concrete input syntax, and transformation rules. Before doing that however, it might be more useful to first define a small language to help with explanations. Look at Figure 5, which is the small subset of PSDL used in designing the initial SDE for CAPS [Ref. 14]. This

subset should be small enough to provide a good example and large enough to get the reader a little accustomed to PSDL. The complete PSDL grammar is in Appendix A.

```
psdl
   = {component}
component
   = data_type | operator
data_type
   = "type" id type_spec
operator
   = "operator" id operator_spec
type_spec
   = "specification" [type_decl] "end"
operator_spec
   = "specification" {interface} "end"
interface
   = attribute [reqmts_trace]
attribute
   = input | output
input
   = "input" type_decl
output
   = "output" type_decl
reqmts_trace
   = "by requirements" id_list
type_decl
   = id_list ":" id
id_list
   = id {"," id}
id
   = letter {alphanumeric}
alphanumeric
   = letter | digit
letter
   = "a..z" | "A..Z" | "_"
digit
   = "0..9"
```

Figure 5. Subset of PSDL Grammar.

This may look somewhat confusing at first, but is really rather simple after a few minutes worth of inspection. Brace brackets ({ }) indicate zero or more iterations, while square brackets ([]) indicate zero or one iteration. The pipe (|) and a small circle with a plus sign in it (used in the next figure) indicates exclusive-or. One might read this as, a psdl prototype is composed of zero or more components, which are each either a data_type or an operator. A data_type is composed of the literal "type", an id, and a type_spec, and so on.

Figure 6 depicts the same information in a form that may be more useful. The only true terminal nodes on this tree are the ones surrounded in quotes. Other nodes that are leafs are further expanded somewhere else in the tree. The thing to remember is that much like a compiler, SynGen will search the tree until it correctly reaches a terminal node ensuring that the pieces of the program match the syntactic constraints of the language.

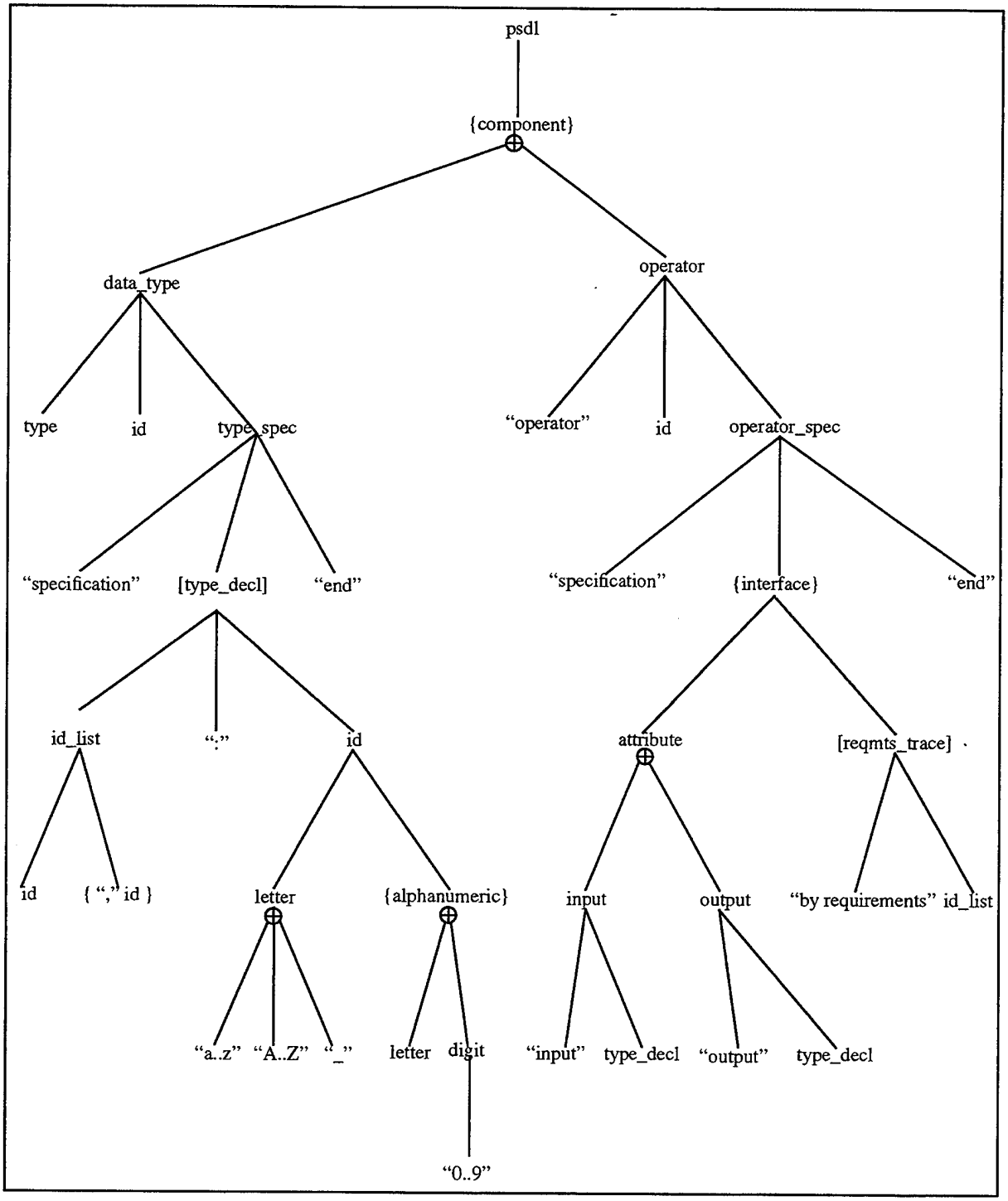


Figure 6. Subset of PSDL Grammar.

1. Abstract Syntax Rules

The abstract syntax is the core of the editor and is defined as a set of grammar rules. Anything constructed or modified within the editor will be represented by a derivation tree that is built based on the grammar. It maintains how legal tokens and productions are allowed. Figure 7 shows one of several possibilities for representing the abstract rules for the grammar previously defined. Not all of the productions were specified to keep its size more manageable.

```
root prototype;
 /*****          PARTIAL PSDL LEXEMES  ****/
BOOLKW
  :  <"BOOLEAN" >
  |  <"boolean" >;
TRUEKW
  :  <"TRUE" >
  |  <"true" >;
FALSEKW
  :  <"FALSE" >
  |  <"false" >;
IDENTIFIER
  : IdentLex<[a-zA-Z][a-zA-Z_0-9]* >;
/*****          PARTIAL PSDL PRODUCTIONS ****/
prototype
  :  Prot(psdl_components);
list psdl_components;
psdl_components
  :  PsdlNil()
  |  PsdlPair(component psdl_components);
component
  :  ComponentNull()
  |  Data(id type_spec)
  |  Op(id operator_spec);
operator_spec
  :  OpSpec(interface_list);
type_spec
  :  TypeSpec(optional_type_declaration);
optional list optional_interface;
interface_list
  :  InterFaceNil()
  |  InterFaceList(interface interface_list);
interface
  :  InterfaceNull()
  |  Interface(attribute optional_requirements);
attribute
  :  Input(type_decl)
  |  Output(type_decl);
optional optional_requirements;
optional_requirements
  :  OptReqmtsTraceNull()
  |  OptReqmtsTracePrompt()
  |  OptReqmtsTrace(id_list);
type_decl
  :  TypeDecl(id_list id);
optional optional_type_declaration;
optional_type_declaration
  :  OptTypeDeclNull()
  |  OptTypeDecl(id_list id);
list id_list;
id_list
  :  IdNil()
  |  IdPair(id id_list);
id
  :  IdNull()
  |  Id(IDENTIFIER);
```

Figure 7. Abstract Syntax Declarations.

Before proceeding, some of SynGen's terminology should be explored. These first three definitions are recursively defined and will make a beginner's head hurt until s/he has gone through several examples. A *phylum* is a set of terms. A *term* is the result of applying a k-ary operator to k terms of the appropriate phyla (plural of phylum). A *k-ary operator* is a constructor-function mapping k terms to a term [Ref. 15]. The phylum associated with a non-terminal is the set of derivation trees that can be derived from it. Those derivation trees (called terms) are derived by going through the productions identified by the operators. We will see an example of this shortly. Phylum declarations are either *productions* or *lexemes*. The legal production declarations allowed, which are described in the abstract syntax rules, take on a form something like:

phylum-name : operator_name (phylum$_1$, phylum$_2$, ... , phylum$_k$);

where:

phylum_name is the particular phylum this production applies to,

operator_name is any legal identifier that refers to a particular production, and

phylum$_i$ represents a non terminal of the grammar.

The legal tokens allowed, defined by the lexeme declarations within the abstract syntax rules, take on a form such as:

phylum-name : lexeme_name < regular_expression >;

where:

phylum_name is the particular phylum this lexeme belongs to,

lexeme_name is used in the definition of the concrete input grammar, and

regular_expression is the description of the token.

A few more definitions are required at this point. Each phylum contains a unique term called its *completing term* and *placeholder term.* While the same term can be both, there are differences between them that must be discussed. The completing term is used to construct the derivation tree's default representation. Whenever there is an unexpanded occurrence of a phylum in the derivation tree, there will be an instance of the appropriate completing term. The placeholder term identifies where subterms can be inserted or swapped. This will become more apparent when the transformation and unparsing rules are covered.The first operator declared for a phylum is that phylum's *completing operator.* The completing operator is used to build the completing term; it is always the first operator in the completing term. The rest of the

completing term depends on whether it is ordinary, a list, or optional.

With *ordinary phyla* (not list nor optional), both completing and placeholder terms are created by applying the completing operator to the completing terms of its arguments. Nullary terms are stopping points and their parenthesis are optional. For example, the completing term for type_name is TypeName(IdNull). The completing operator of type_name is TypeName, and it has only one argument. The completing operator of its argument, id, is IdNull, which is a nullary operator. See Figure 8 below to show the components affected.

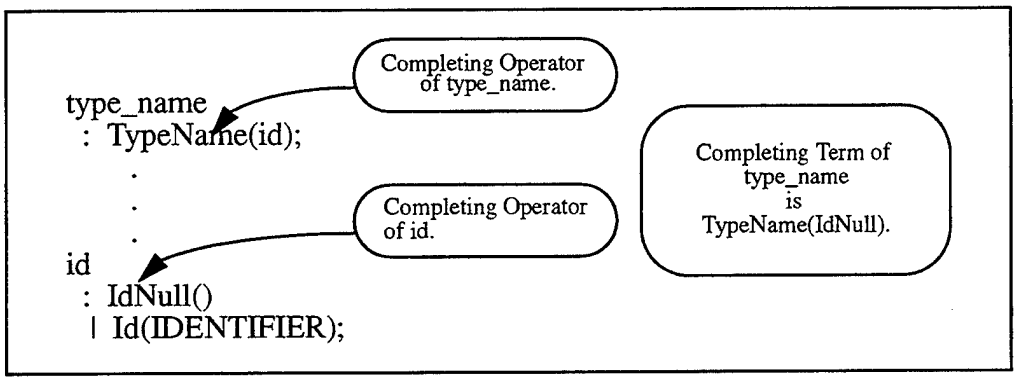


Figure 8. Components of a Completing Term.

If the phylum is of *non-optional list phyla*, the completing term and placeholder term are also equal and are built by applying its binary operator to the completing term of its left argument phyla and to the list's nullary operator [Ref. 13], resulting in a singleton list. This makes more sense when it is realized that all list phyla must have two argument phyla, where the first is another phylum and the second is the list itself (all lists are right recursive). For example, the completing term for id_list is IdPair(IdNull, IdNil). The binary operator is "IdPair", the completing term of its left argument is "IdNull", and the list's nullary operator is "IdNil". A look back to Figure 7 will help follow this reasoning.

Attempting a combination involves using both rules. For example, the completing term of type_decl is TypeDecl(IdPair(IdNull, IdNil), TypeName(IdNull)). Noting that type_decl is ordinary, you combine the completing operator, TypeDecl, to the completing terms of its arguments; id_list and type_name. Since their completing terms are already known (from the last two paragraphs), it now just a matter of combining them.

When dealing with optional phyla, the completing term and placeholder term are different. For *optional, non-list phyla*, the completing term is built from its first nullary operator

while the placeholder term is built from the first operator after the placeholder term. For example, the completing term of optional_requirements is OptReqmtsTraceNull. The placeholder term is OptReqmtsTracePrompt. These two nullary terms are in the optional phyla because the first will result in nothing being displayed in the editor while the second will allow the editor to display a prompt. This is covered in the unparsing rules.

With the last category, *optional list phyla*, the completing term is the nullary operator and the placeholder is the same as for the non-optional list phyla covered above.

2. Attribute Rules

So far, the underlying structure of the editor has been described. The next question that comes to mind is how all of the syntactic and semantic checking /analysis is done. This involves various parts of the derivation tree knowing about and having some sort of an understanding of other parts of the tree. In other words, information must be passed up/down and back/forth within the derivation tree. It is accomplished with attributes and attribute equations.

The attribute rules make up an *attribute grammar*, which is a context-free grammar (CFG) that is extended by the use of attributes which are attached to non-terminals and defined by *attribute equations* [Ref. 13]. There are several kinds of attributes available to SynGen. While all are covered in the references [Ref. 13, 15], only three will be discussed here as they apply directly to the SDE within CAPS.

The editor designer can use *local attributes*, which are associated with a particular production. They are declared with the reserved word, "local" and are declared with the production attribute equations. The format of a local attribute declaration is as follows.

local attribute_type attribute_ name;

where:

local is a keyword,

attribute_type is any predefined type,

attribute_name is any valid identifier.

The attributes that are associated with phyla, instead of productions, are broken into two mutually exclusive (disjoint) sets: synthesized attributes and inherited attributes.

Synthesized attributes are attributes that are built up. They are attributes whose values are propagated to the left in attribute equations and up the attributed tree. An example will be given in a moment.

Inherited attributes are attributes that are passed down. They are attributes whose values are propagated to the right in attribute equations and down the attributed tree. The attribute declaration format is as follows.

a_phylum { synthesized attribute_phylum attribute_name;
inherited attribute_phylum attribute_name;};

where:

a_phylum is the phylum or phyla (comma separated) that will have the attribute,
synthesized is the keyword to declare a synthesized attribute (syn is OK),
inherited is the keyword to declare an inherited attribute (inh is OK),
attribute_phylum is the attributes type and can be built-in or used defined,
attribute_name is any valid identifier.

It should also be mentioned that the two attributes can be in either order and can be in separate declarations if desired.

Attribute equations, which are used to assign values to attributes and evaluate values of attributes, take on a slightly different form depending on if they are for synthesized or inherited attributes. The synthesized attribute equation and inherited attribute equation respectively, are as follows.

phylum_name.attribute_name = subordinate_phylum_value;
subordinate_phylum_value = phylum_name.attribute_name;

where:

phylum_name is the phylum that owns the attribute,
attribute_name is the attribute owned by phylum_name,
subordinate_phylum_value is a little bit tricky. It can be any value consistent with the attribute's type including another attribute, the same attribute from another phylum, or a function call, and is associated with a lower level of the attributed tree.

In some abstract syntax declarations (a list for example), the phylum_name is mentioned more than once. Because of this, additional notation must be introduced. The phylum_name by itself implies its first occurrence. Another way to depict this is with $$. The second occurrence, would be indicated as phylum_name$2. Then, every subsequent occurrence gets the number after the $ symbol incremented by one.

As an example, review the abstract syntax presented in Figure 7 above and notice that a prototype is simply a list of components. Recall that all lists are binary and right recursive. A component is either a data_type denoted with the phylums id and type_spec, an operator with the phylums id and operator_spec, or null. The id simply identifies the name of that component, whether it be a type or operator. Obviously, we do not want two components to have the same name, regardless of whether they are types or operator.

In the PSDL Editor, the graphic editor allows the designer to draw and name operators. This information is propagated to the prototype when exiting back to the SDE. That means the operators are already defined when you enter the SDE. It might be beneficial to notify the designer when trying to assign a new data_type with the same name as an existing operator. After the designer enters three operators (Op1, Op2, Op3) in the graphic editor and returns to SDE, the attributed tree would look similar to Figure 9 (attributes are bold typed).

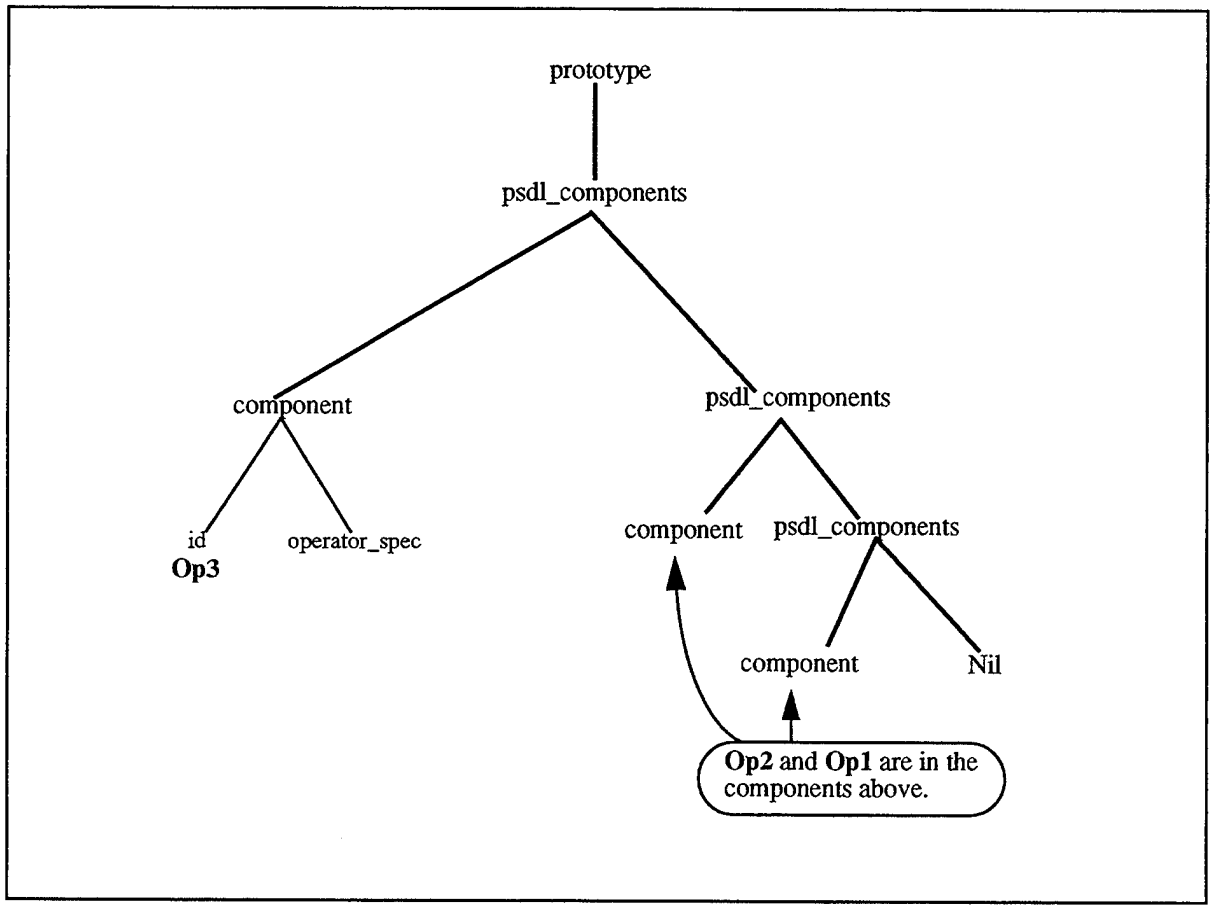


Figure 9. Partial Derivation Tree after returning from Graphic Editor.

The information about operator names must be sent to where data_type names are declared. Another way of saying this is that operator id attributes must be synthesized and then inherited by the data_type. Currently, there are only three components, and all three are operators. When data_types are added, the list of psdl_components will grow containing both operators and data_types.

The first thing that must be done is to collect up all the operator id's. Because they exist at the id phylum, that is where we will put the synthesized attribute declaration. That attribute will be passed up to the component phylum so it too will need an id attribute. And because they will be collected up into a set at the psdl_component level and subsequently passed to the prototype level, we must define a structure to hold a set of id's and declare attributes of that type at the psdl_component phylum and the prototype phylum. Of course the skeleton attribute grammar must be in place for this to be inserted into. It looks like the production portion of the abstract syntax declarations displayed earlier. See Figure 10 to understand what is required in collecting all operator id's at the prototype phylum. All additions to the abstract syntax are in bold type.

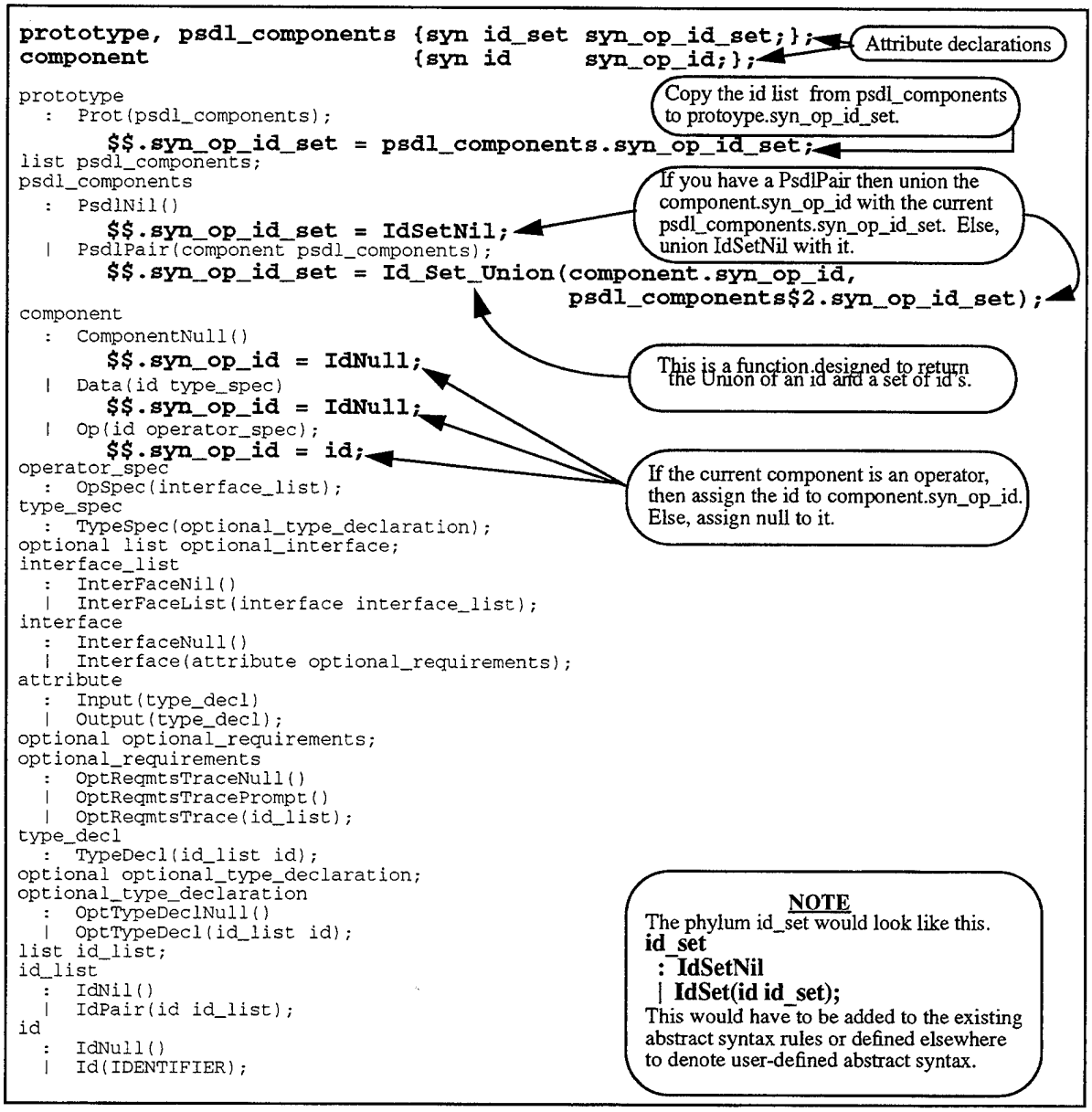


Figure 10. Partial Abstract Grammar showing Synthesized Attributes.

When studying the changes made for attribution, notice the way upper and lower case is used. This is very important to help keep from getting confused when looking at larger specifications such as that for psdl.

Figure 9 was duplicated in Figure 11 below except that the newly declared synthesized attributes are attached to the appropriate phylum.

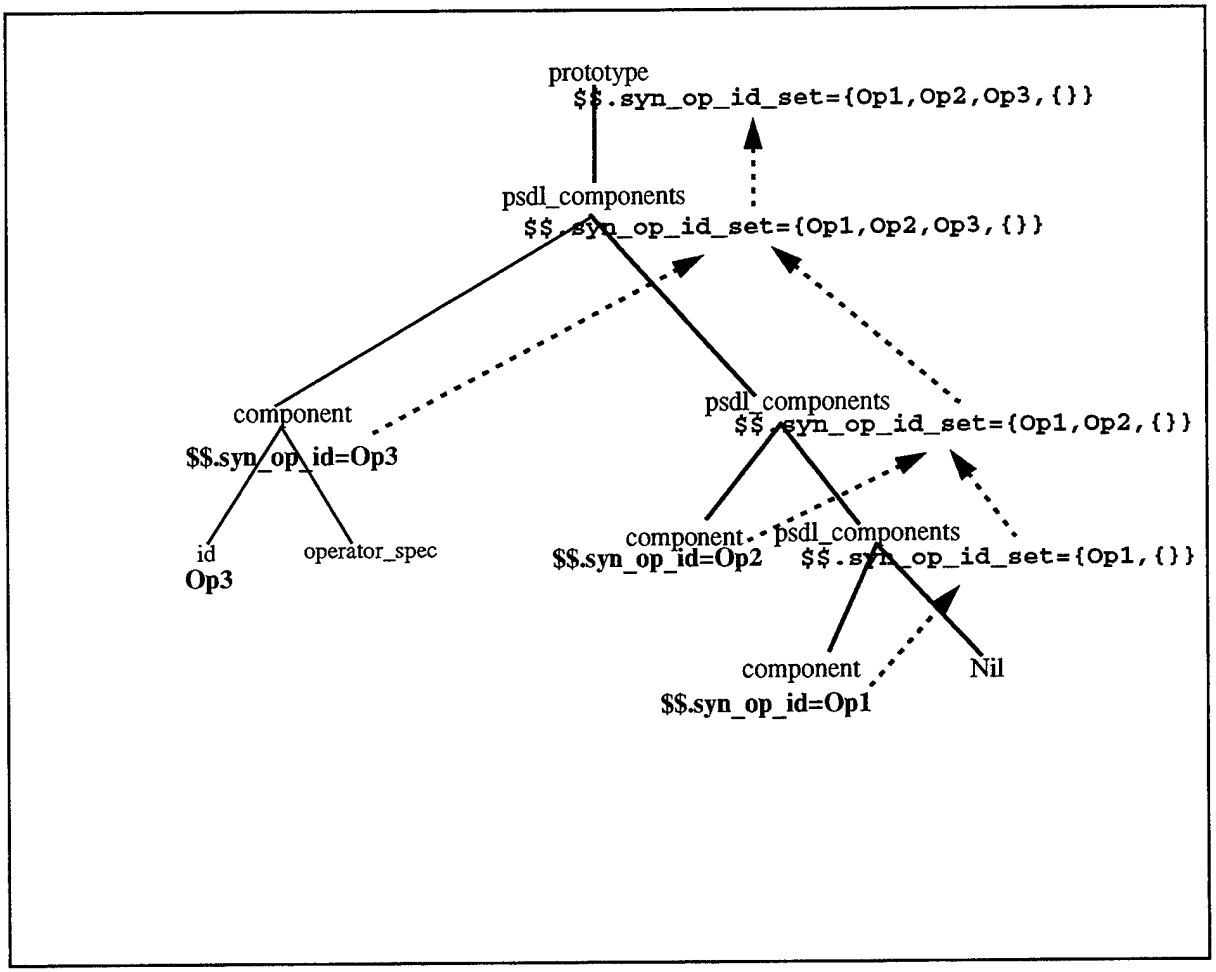


Figure 11. Partial Derivation Tree reflecting Synthesized Attributes.

Now we have all the operator id's at the prototype phylum. The next thing to do is to send that list of operator names back down to the component phyla so that whenever a data_type is about to be identified, the editor can make sure the id doesn't already exist. The nullary phyla do not have need of it, as they have nothing to compare it against. It is much simpler and is essentially the same process, only in reverse; it is inheritance, and is shown in Figure 12. The only possible tricky part is when dealing with lists. At phylum psdl_components, the list must be passed down to the current component and to the rest of the list (psdl_components$2).

```
prototype, psdl_components, component {inh id_set inh_op_id_set;};
                          (This attribute was built in Fig 10&11.)
prototype
   :  Prot(psdl_components);
        $$.inh_op_id_set = $$.syn_op_id_set;
        psdl_components.inh_op_id_set = $$.inh_op_id_set;
list psdl_components;
psdl_components
   :  PsdlNil()
   |  PsdlPair(component psdl_components);
        component.inh_op_id_set          = $$.inh_op_id_set;
        psdl_components$2.inh_op_id_set = $$.inh_op_id_set;
 component
   :  ComponentNull()
   |  Data(id type_spec)
   |  Op(id operator_spec);
operator_spec
   :  OpSpec(interface_list);
type_spec
   :  TypeSpec(optional_type_declaration);
optional list optional_interface;
interface_list
   :  InterFaceNil()
   |  InterFaceList(interface interface_list);
interface
   :  InterfaceNull()
   |  Interface(attribute optional_requirements);
attribute
   :  Input(type_decl)
   |  Output(type_decl);
optional optional_requirements;
optional_requirements
   :  OptReqmtsTraceNull()
   |  OptReqmtsTracePrompt()
   |  OptReqmtsTrace(id_list);
type_decl
   :  TypeDecl(id_list id);
optional optional_type_declaration;
optional_type_declaration
   :  OptTypeDeclNull()
   |  OptTypeDecl(id_list id);
list id_list;
id_list
   :  IdNil()
   |  IdPair(id id_list);
id
   :  IdNull()
   |  Id(IDENTIFIER);
```

Figure 12. Partial Abstract Grammar showing Inherited Attributes.

The problem now is that the component has the list but doesn't know what to do with it. This answer to this involves the use of local attributes, rules defining error attributes, and possibly auxiliary functions. The functions for defining error attributes and auxiliary functions can be with the attribute rules or within another file (more common).

First, declare a local BOOL(predefined in SSL) attribute called duplicate_id_error within the component phylum under the Data operator. This will be set to true if the current data_type id already exists within the inh_op_id_set. Then a function must be written to search the list for the data_type id. Figure 13 shows where this would be added.


```
prototype
   :  Prot(psdl_components);
list psdl_components;
psdl_components
   :  PsdlNil()
   |  PsdlPair(component psdl_components);
component
   :  ComponentNull()
   |  Data(id type_spec)
         local BOOL duplicate_id_error;
         duplicate_id_error = Id_In_Op_Id_Set(id, $$.syn_op_id_set);
         local STR duplicate_id_msg;
         duplicate_id_msg = (duplicate_id_error)
                           ? "\n"
                            #" This id is already an operator id."
                           : "";
   |  Op(id operator_spec);
operator_spec
   :  OpSpec(interface_list);
type_spec
   :  TypeSpec(optional_type_declaration);
optional list optional_interface;
interface_list
   :  InterFaceNil()
   |  InterFaceList(interface interface_list);
interface
   :  InterfaceNull()
   |  Interface(attribute optional_requirements);
attribute
   :  Input(type_decl)
   |  Output(type_decl);
optional optional_requirements;
optional_requirements
   :  OptReqmtsTraceNull()
   |  OptReqmtsTracePrompt()
   |  OptReqmtsTrace(id_list);
type_decl
   :  TypeDecl(id_list id);
optional optional_type_declaration;
optional_type_declaration
   :  OptTypeDeclNull()
   |  OptTypeDecl(id_list id);
list id_list;
id_list
   :  IdNil()
   |  IdPair(id id_list);
id
   :  IdNull()
   |  Id(IDENTIFIER);
```

Figure 13. Partial Abstract Grammar showing Inherited Attributes.

Most of SSL including the auxiliary functions that can be written are very much like the C language. The "!" symbol, for example, means Not and the "(expression) ? :" is a conditional expression. Figure 14 below shows what the Id_In_Op_Id_Set would look like as an auxiliary function. Immediately following that is Figure 15 which represents the newly declared inherited attributes attached to the appropriate phylum.

```
BOOL Id_In_Op_Id_Set(id data_type_name, id_set  operator_names) {
  with(operator_names) (
     IdSetNil:   false,
     IdSet(head, tail):
    (head == data_type_name)
     ? true
      : Id_In_Op_Id_Set(data_type_name, tail)
  )
};
```

Figure 14. Auxiliary Function.

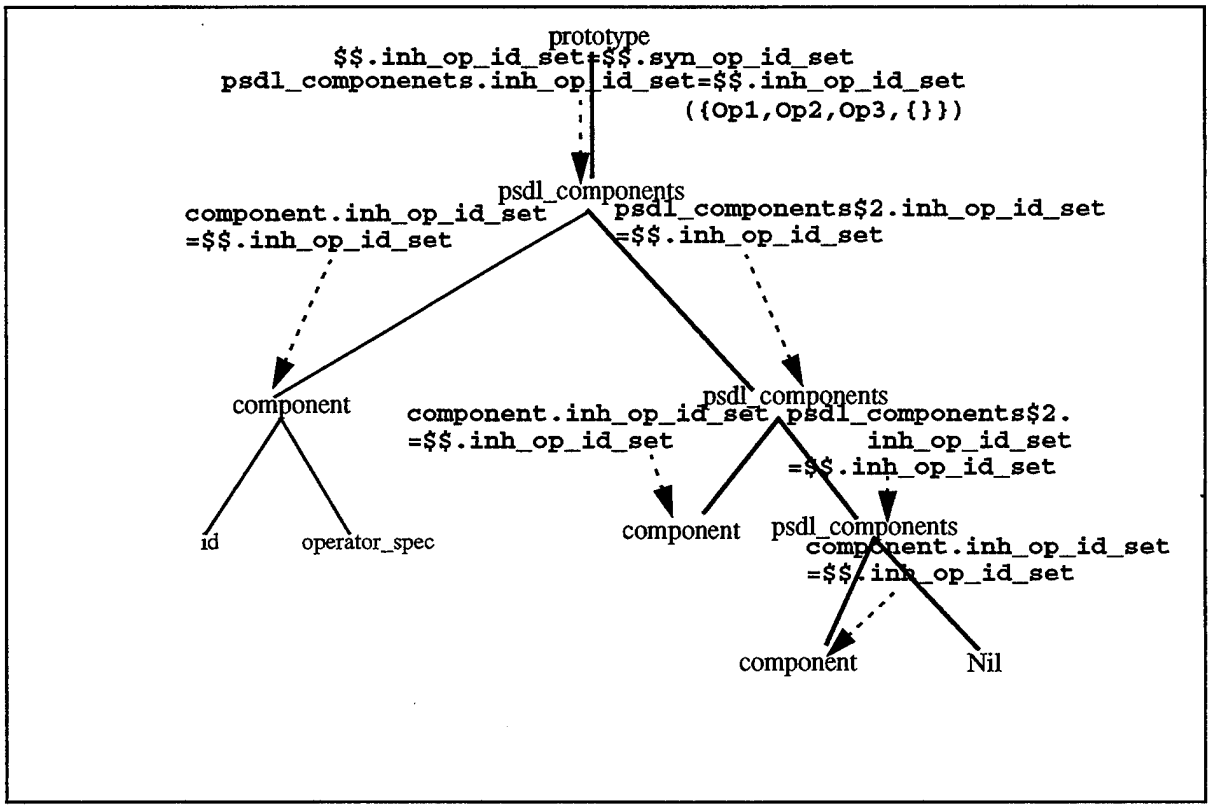


Figure 15. Partial Derivation Tree reflecting Inherited Attributes.

In Figure 16, all of the attribute modifications are reflected at once. There are no big surprises except for combining synthesized and inherited attributes in the same declaration. The next section will concentrate on what the user sees, including notification of an error identified in the attribute rules.

```
prototype, psdl_components {syn id_set syn_op_id_set;
                            inh is_set inh_op_id_set;};
component                  {syn id     syn_op_id;
                            inh id_set inh_op_id_set;};

prototype
  : Prot(psdl_components);
      $$.syn_op_id_set = psdl_components.syn_op_id_set;
      psdl_components.inh_op_id_set = $$.inh_op_id_set;
list psdl_components;
psdl_components
  : PsdlNil()
      $$.syn_op_id_set = IdSetNil;
  | PsdlPair(component psdl_components);
      $$.syn_op_id_set = Id_Set_Union(component.syn_op_id,
                                      psdl_components$2.syn_op_id_set);
      component.inh_op_id_set          = $$.inh_op_id_set;
      psdl_components$2.inh_op_id_set = $$.inh_op_id_set;
component
  : ComponentNull()
      $$.syn_op_id = IdNull;
  | Data(id type_spec)
      $$.syn_op_id = IdNull;
      local BOOL duplicate_id_error;
      duplicate_id_error = Id_In_Op_Id_Set(id, $$.syn_op_id_set);
      local STR duplicate_id_msg;
      duplicate_id_msg = (duplicate_id_error)
                         ? "\n"
                          #" This id is already an operator id."
                         : "";
  | Op(id operator_spec);
      $$.syn_op_id = id;
operator_spec
  : OpSpec(interface_list);
type_spec
  : TypeSpec(optional_type_declaration);
optional list optional_interface;
interface_list
  : InterFaceNil()
  | InterFaceList(interface interface_list);
interface
  : InterfaceNull()
  | Interface(attribute optional_requirements);
attribute
  : Input(type_decl)
  | Output(type_decl);
optional optional_requirements;
optional_requirements
  : OptReqmtsTraceNull()
  | OptReqmtsTracePrompt()
  | OptReqmtsTrace(id_list);
type_decl
  : TypeDecl(id_list id);
optional optional_type_declaration;
optional_type_declaration
  : OptTypeDeclNull()
  | OptTypeDecl(id_list id);
list id_list;
id_list
  : IdNil()
  | IdPair(id id_list);
id
  : IdNull()
  | Id(IDENTIFIER);
```

Figure 16. Partial Abstract Grammar Showing All Attributes.

3. Unparsing Rules

As just mentioned, the editor can now determine if the user is attempting to assign an operator name to a data_type that already exists. In fact, there is a local attribute that contains the message to be displayed to the user if this situation arises. It is up to the unparsing rules to display the string defined in the attribute rules. The unparsing rules define everything that the user sees. They also controls which nodes of the abstract syntax tree are selectable and which productions are editable. The form in which unparsing rules appear is as follows.

phylum : operator [unparsing_syntax];

where:

operator and phylum match the abstract syntax one for one, and the

unparsing_syntax is the specification for how that part of the tree is displayed and is broken up into a left side and a right side, coinciding with the productions of the abstract syntax tree. The left and right side are divided by a ":", denoting immutable text, or a "::=", denoting mutable text.

There is another symbol called the *selection symbol* which represents the position of each phylum occurrence. Which one is used depends on whether or not that phylum occurrence should be a resting place for the editor. The "@" selection symbol specifies that the phylum is a resting place while the "^" selection symbol specifies that it is not. The designer of an editor must keep in mind that almost all phyla exist once on the right hand side (RHS) and once on the left hand side (LHS). Therefore, if a phylum is or is not to be selectable, both occurrences must be considered. It makes sense to use one side or the other to determine selectability. The examples here will use the LHS, which means the selection symbols on the RHS will be "^".

Control characters are allowed to help control the display of the screen. Table 1 lists them, while only three will be used here; %t for tab, %b for back tab, and %n for newline.

Command	Action
%t	move the left margin one indentation unit to the right
%b	move the left margin one indentation unit to the left
%n	newline, return to the current left margin
%1	return to the current left margin and overprint
%1	move to column one of the same line and overprint
%T	move right to the next tab stop
%M(c)	move right to column c, where c is a positive integer
%o	optional newline, return to the current left margin
%c	same as %o, but either all or no %c in a group are taken
%{	beginning of an unparsing group
%}	end of an unparsing group
%[	same as %t%{
%]	same as %}%b
%S(style-name	enter the named style
%S)	revert to the previous style
%%	display a %

Table 1. SSL Display Formatting Commands.

The unparsing rules for the PSDL subset presented earlier are shown in Figure 17. Notice that whether a phylum is selectable is determined by its LHS occurrence. Also notice some consistency was maintained with respect to placement of formatting commands; whether at the beginning of the appropriate line or the end of the preceding line.

All of keywords or phrases that were identified in the PSDL grammar seemed to disappear in the other rules. Here, however, they have returned and are displayed in the appropriate places.

One last thing to note is the specification for a data_type where the user is now notified about "duplicate_id_msg" errors. The string variable, "duplicate_id_msg", is always displayed at the Data production. The difference is that, if an error exists, the string variable will contain a message stating that fact, and if an error does not exist, the variable string will be a nullstring.

```
prototype
    : Prot              [@:^};
psdl_components
    : PsdlNil           [@:]
    | PsdlPair          [@:^["%n"]^];
component
    : ComponentNull     [^:"%n[component]"]
    | Op                [^:"%nOPERATOR "^^]
    | Data              [^:"%nTYPE "^ duplicate_id_msg ^]
operator_spec
    : OpSpec            [^:"%nSPECIFICATION%t"^"%b%nEND"]
type_spec
    : TypeSpec          [^:"%nSPECIFICATION%t"^"%b%nEND"];
optional_interface_list
    : InterFaceNil      [@:]
    | InterFaceList     [@:^[]@];
interface
    : InterfaceNull     [@::="<interface>"]
    | Interface         [@:^^];
attribute
    : EmptyAttr         [^:"%n{interface}"]
    | Input             [^:"%nINPUT%n%t"^"%b"]
    | Output            [^:"%nOUTPUT%n%t"^"%b"]
optional_requirements
    : ReqmtsTraceNone   [@:]
    | ReqmtsPrompt      [@:"%n{requirements}"]
    | ReqmtsTrace       [@:"%nBY REQUIREMENTS%t%n"^"%b"];
type_decl
    : TypeDecl          [^:^" : "^]
optional_type_declaration
    : OptTypeDeclNull   [@:"%n{optional type declaration}"]
    | OptTypeDecl       [@:"%n"^" : "^];
id_list
    : IdNil             [^:"<identifier"]
    | IdPair            [^:^];
id
    : IdNull            [@::="<identifier>"]
    | Id                [@::=^];
```

Figure 17. Unparsing Rules for Abstract Syntax.

4. Transformation Rules

Transformation rules allow for the restructuring of objects. While transformations can include various kinds of computations, this section will focus on the more general case of template insertion. This can only happen when the selection is a placeholder. The format of a typical template insertion is as follows.

transform phylum on transformation_name pattern : expression;

where:

transform is a keyword identifying a transformation,

phylum identifies the phylum that this transformation can be done on,

on is another keyword,

transformation_name is the descriptive identifier in the help window which the user selects with the left mouse to activate the transformation,

pattern is either the same as the phylum or some appropriate sub-phylum, and expression is the template that is inserted where the pattern was.

The transformation rules for the PSDL subset is displayed in Figure 18.

```
transform component
    on "type"
        <component> :
        Data(<id>,<type_spec>),
    on "operator"
        <component> :
        Op(<id>,<operator_spec>);

transform attribute
    on "input"
        <attribute> :
        Input(<input>,<optional_requirements>),
    on "output"
        <attribute> :
        Output(<output>,<optional_requirements>);

transform optional_requirements
    on "enter_requirements"
        <optional_requirements> :
        ReqmtsTrace(<id_list>);

transform optional_type_declaration
    on "enter_declaration"
        <optional_type_declaration> :
        OptTypeDecl(<id_list>,<type_name>);
```

Figure 18. Transformation Rules for Abstract Syntax.

5. Concrete Rules

The concrete rules, or concrete input syntax, is needed to provide the editor with the ability to do text input and to load a pre-existing program into the editor.

It allows for text editing by parsing and then translating the string to the corresponding term of the abstract syntax. While the subterm is being edited, it can be any string. Once the user attempts to move to a different part of the structure, it is parsed. If syntactically correct, it replaces the term in the abstract syntax. If it is not correct, the cursor is positioned at the beginning of the string and an error message is displayed.

It allows for editing existing programs by specifying rules that build a tree structure coinciding with the abstract syntax rules previously defined. This means that as a minimum, the concrete rules must specify a phylum in the input syntax to be associated with all phylum of the abstract syntax which are to be edited as text. If pre-existing programs are to be re-edited, the whole language must be supported.

There are several parts to the concrete rules, most of which is similar to what has already been covered above. The rules include attribute declarations which have a form such as:

phylun_name {type_attribute attribute_phylum attribute_name};

where:

phylum_name is a phylum within the input grammar,

type_attribute is either synthesized or inherited,

attribute_phylum is the attributes type and can be built-in or used defined,

attribute_name is any valid identifier.

Entry declarations are required to create a correspondence between the entry points within the input syntax and the appropriate selection within the abstract syntax. Their format is as follows:

p ~ P.t;

where:

p is a phylum such that the current selected component is of that phylum, and

P is an input phylum such that the input is parsed to see if it is of that phylum, and

t is a synthesized attribute that if the input is of phylum P, is used to replace the current selected component [Ref. 13]. In other words, if the two phylums match, the attribute goes in the derivation tree.

The input syntax must have tokens, so there are also lexemes in the concrete rules. There is one rule for each keyword or token. All white space is ignored during the parsing of the input program. The format of a lexical phylum declaration is the same as for the abstract syntax rule. For convenience, it is repeated below.

phylum-name : lexeme-name regular-expression ;

Last, but certainly not least, is the productions of the input grammar or *parsing declarations*. A slight difference from what was explained previously is the use of the $. In this context, the phylum appended with $1 means the first occurrence and $2 is the second occurrence, and so on. The productions of the input syntax can be ambiguously specified in which case there must be disambiguating precedence rules. The general form is shown next.

phylum_name ::= (phylum_token_comb) {LHS = RHS;}

where:

phylum_name is self explanatory,

::= is the symbol to separate the LHS phylum name from the RHS symbols,

phylum_token_comb is a phylum, token, or a combination of both which is parsed,

LHS=RHS is the assignment of some value to the attribute of the phylum_name.

On the following page is Figure 19, which contains the concrete rules for the subset of PSDL that has been the ongoing base throughout this chapter. The use of inherited attributes was not covered in this section as it is somewhat more detailed. It is covered extensively in references [Ref. 13, 15], however, for the interested reader. Also, the concrete rules for PSDL in the CAPS SDE do use inherited attributes to deal with the fact that the productions build lists in reverse order.

Hopefully several things have been accomplished in this chapter. First, the reader should have a better appreciation for how powerful the SynGen really is. Second, s/he should have a little better grasp of what PSDL is and how it is structured. In the next chapter, a small sample application is introduced. In that section, particularly when dealing directly with SDE, the reader should be continually reminded of the rules discussed in this chapter. Significant aspects of the first four rules are in that sample application. Everything shown to the user was specified in the unparsing rules. All warning and error messages are defined and discovered within the attribute rules and auxiliary functions. All transformations in the help window (bottom) of the SDE are specified in the transformation rules. While not evident in the sample application, bringing an existing prototype into the SDE requires the concrete rules. And finally, none of the above would be possible without the derivation tree representation of the prototype itself, specified through the abstract syntax rules.

```
/* Attribute Declarations */
PROTOTYPE                  {synthesized prototype t;};
PSDL_COMPONENTS            {synthesized psdl_components t:};
COMPONENT                  {synthesized component t;};
OPERATOR_SPEC              {synthesized operator_spec t;};
TYPE_SPEC                  {synthesized type_spec t;};
OPTIONAL_INTERFACE_LIST    {synthesized optional_interface_list t;};
INTERFACE                  {synthesized interface t;};
ATTRIBUTE                  {synthesized attribute t;};
OPTIONAL_REQUIREMENTS      {synthesized optional_requirements t;};
TYPE_DECL                  {synthesized type_decl t;};
OPTIONAL_TYPE_DECLARATION {synthesized optional_type_declaration t;};
ID_LIST                    {synthesized id_list t;};
ID                         {synthesized id t;};
/* Entry Declarations */
prototype                  ~ PROTOTYPE.t;
psdl_components            ~ PSDL_COMPONENTS.t;
component                  ~ COMPONENT.t;
operator_spec              ~ OPERATOR_SPEC.t;
type_spec                  ~ TYPE_SPEC.t;
optional_interface_list    ~ OPTIONAL_INTERFACE_LIST.t;
interface                  ~ INTERFACE.t;
attribute                  ~ ATTRIBUTE.t;
optional_requirements      ~ OPTIONAL_REQUIREMENTS.t;
type_decl                  ~ TYPE_DECL.t;
optional_type_declaration ~ OPTIONAL_TYPE_DECLARATION.t;
id_list                    ~ ID_LIST.t;
id                         ~ ID.t;
/* Lexemes Same as before and not specified again. */
/*Parsing Declarations */
PROTOTYPE                  ::= (PSDL_COMPONENTS)
                               {PROTOTYPE.t = PSDL_COMPONENT.t;}
PSDL_COMPONENTS            ::= (COMPONENT PSDL_COMPONENTS)
                               {PSDL_COMPONENT.t=(COMPONENT.t::PsdlNil;}
                           |   (COMPONENT PSDL_COMPONENTS)
                               {PSDL_COMPONENTS$1.t =
                                 (COMPONENT.t::PSDL_COMPONENTS$2.t);}
COMPONENT                  ::= ()
                               {COMPONENT.t = ComponentNull;}
                           |   ("TYPE" ID TYPE_SPEC)
                               {COMPONENT.t = (Data(ID.t, TYPE_SPEC.t));}
                           |   ("OPERATOR" ID OPERATOR_SPEC)
                               {COMPONENT.t= (Op(ID.t, OPERATOR_SPEC.t));}
OPERATOR_SPEC              ::= ("SPECIFICATION" INTERFACE_LIST "END")
                               {OPERATOR_SPEC.t = INTERFACE_LIST.t;}
TYPE_SPEC                  ::= ("SPECIFICATION" TYPE_DECL "END")
                               {TYPE_SPEC.t = TYPE_DECL.t;}
OPTIONAL_INTERFACE_LIST    ::= ()
                               {OPTIONAL_INTERFACE_LIST.t= InterfaceNil;}
                           |   (INTERFACE OPTIONAL_INTERFACE_LIST)
                               {OPTTIONAL_INTERFACE_LIST$1.t =
                                (INTERFACE.t::OPTIONAL_INTERFACE_LIST$2.t);}
INTERFACE                  ::= ()
                               {INTERFACE.t = InterfaceNull;}
                           |   (ATTRIBUTE)
                               {INTERFACE.t = ATTRIBUTE.t;}
                           |   (ATTRIBUTE OPTIONAL_REQUIREMENTS)
                               {INTERFACE.t = Interface(INTERFACE.t,
                                              OPTIONAL_REQUIREMENTS.t);}
ATTRIBUTE                  ::= ("INPUT" TYPE_DECL)
                               {ATTRIBUTE.t = Input(TYPE_DECL.t);}
                           |   ("OUTPUT" TYPE_DECL)
                               {ATTRIBUTE.t = Output(TYPE_DECL.t);}
OPTIONAL_REQUIREMENTS      ::= ()
                               {OPTIONAL_REQUIREMENTS.t = OptReqTraceNull;}
                           |   (ID_LIST)
                               {OPTIONAL_REQUIREMENTS.t = ID_LIST.t;}
TYPE_DECL                  ::= (ID_LIST ":" ID)
                               {TYPE_DECL.t = TypeDecl(ID_LIST.t, ID.t;}
OPTIONAL_TYPE_DECLARATION ::= ()
                               {OPTIONAL_TYPE_DECLARATION.t = OptTypeDecNull;}
                           |   (ID_LIST ":" ID)
                               {TYPE_DECL.t = TypeDecl(ID_LIST.t, ID.t;}
ID_LIST                    ::= ()
                               {ID_LIST.t = IdListNil;}
                           |   (ID ID_LIST$1)
                               {ID_LIST.t = ID.t::ID_LIST$2.t;}
ID                         ::= ()
                               {ID.t = IdNull;}
                           |   (IDENTIFIER)
                               {ID.t = Id(IDENTIFIER);}
```

Figure 19. Concrete Rules For Abstract Syntax.

IV. SYNTAX-DIRECTED EDITOR (SDE)

As discussed in the previous chapter, the SDE is an editor tailored to PSDL so that it can assist the user in writing correct programs. The power of such a tool is immense and previous work combined with the work of this thesis have only begun to tap its potential. Because the focus of this thesis is on the SDE, it will be covered somewhat extensively here.

This chapter will cover creating an example prototype with the SDE, the constraint checking functionality that is to be added to the SDE, the actual modifications to implement that additional functionality and finally, an example with the modified editor.

A. EXAMPLE OF SDE

1. Simple Tutorial

This section will provide you with the minimal tools necessary to perform software development through rapid prototyping. Your attention will be focused on the PSDL Editor consisting of three separate parts; the Syntax Directed Editor (SDE), the Graph Viewer and the Graphic Editor.

These three parts taken together allow the designer to create the CAPS data flow diagram and PSDL program. They further allow assignment of all timing and control constraints to prototype components which consist of operators and data streams. For a more detailed presentation, see the CAPS Tutorial.

To start CAPS, you simply type "caps". This will start it up in the designer mode. Using the switch "-m" allows CAPS to be started up in the manager mode. We will not use that, as it is beyond the scope of this example. All of the work that is generated under CAPS is saved in a ".caps" directory which is directly under your Home directory. If you do not have it, CAPS will create it for you. The current version of CAPS not only creates the ".caps" directory, if you don't have it, but it also copies two sample prototypes into it. One of these is the one you are about to create. Therefore, to avoid this potential problem, if you don't have a ".caps" directory, create your own. The results of this section will produce a design that looks like Figure 20.

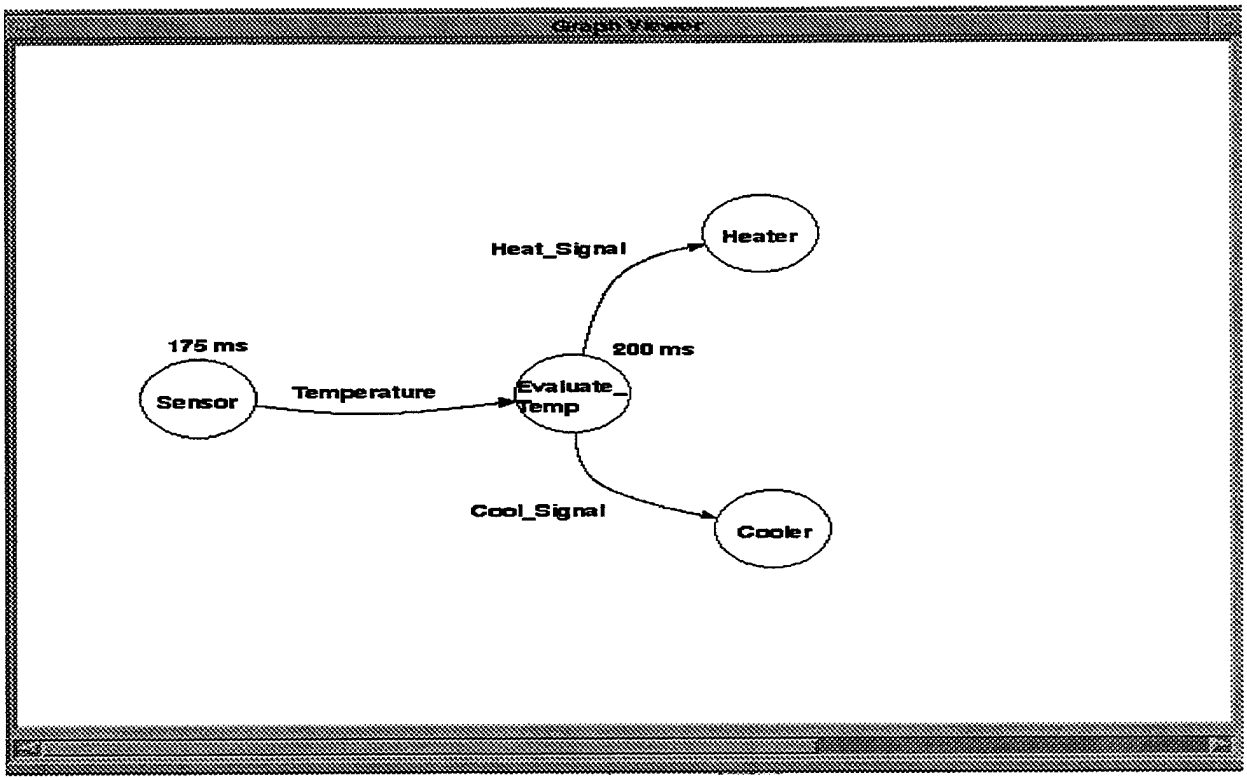


Figure 20. Temp_Controller Prototype displayed in Graph Viewer.

As soon as you start CAPS, you will see an interface like that shown in Figure 21.

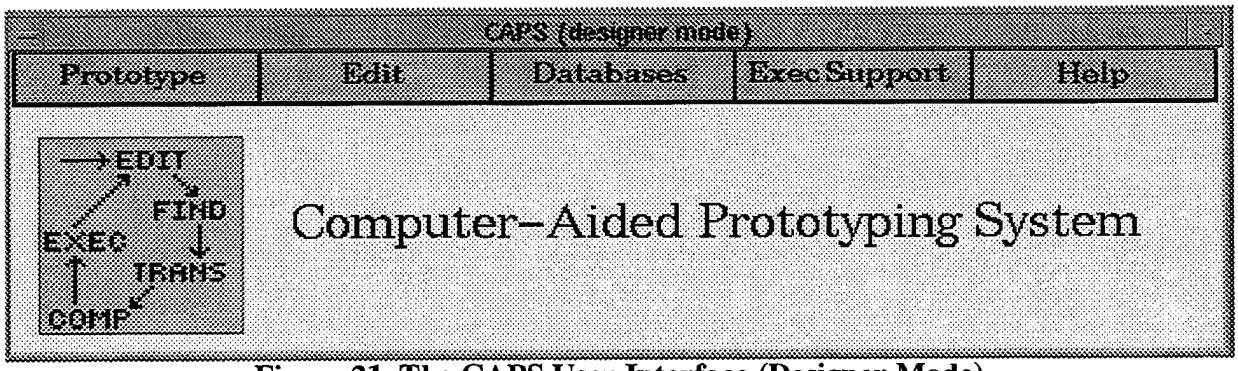


Figure 21. The CAPS User-Interface (Designer Mode).

To select a prototype that already exists, you simply select "Choose" from under the "Prototype" menu, and the available prototypes will be displayed for you to select from. That only selects the prototype. You must then select "PSDL" from under the "Edit" pull down menu to invoke the SDE.

To start a new prototype, pull down the "Prototype" menu from the CAPS designer mode screen and select "New". Do that now and the window displayed in Figure 22 will appear.

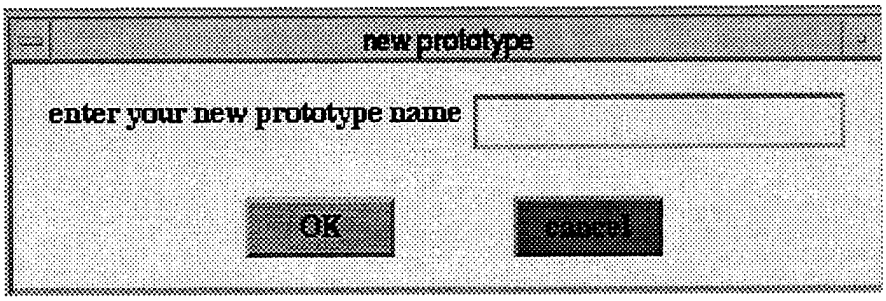


Figure 22. New Prototype Window.

Left click the prototype name box which will cause it to become highlighted. Enter TEMP_CONTROLLER and left click your mouse on the OK button. You must use the mouse to activate the name box and acknowledge ok when finished. A <return> will not work here. In addition, CAPS is case sensitive because the underlying file structure is based on UNIX. Identifiers of two words or more must be connected by an underscore. When you do this, two windows will open up as shown in Figure 23.

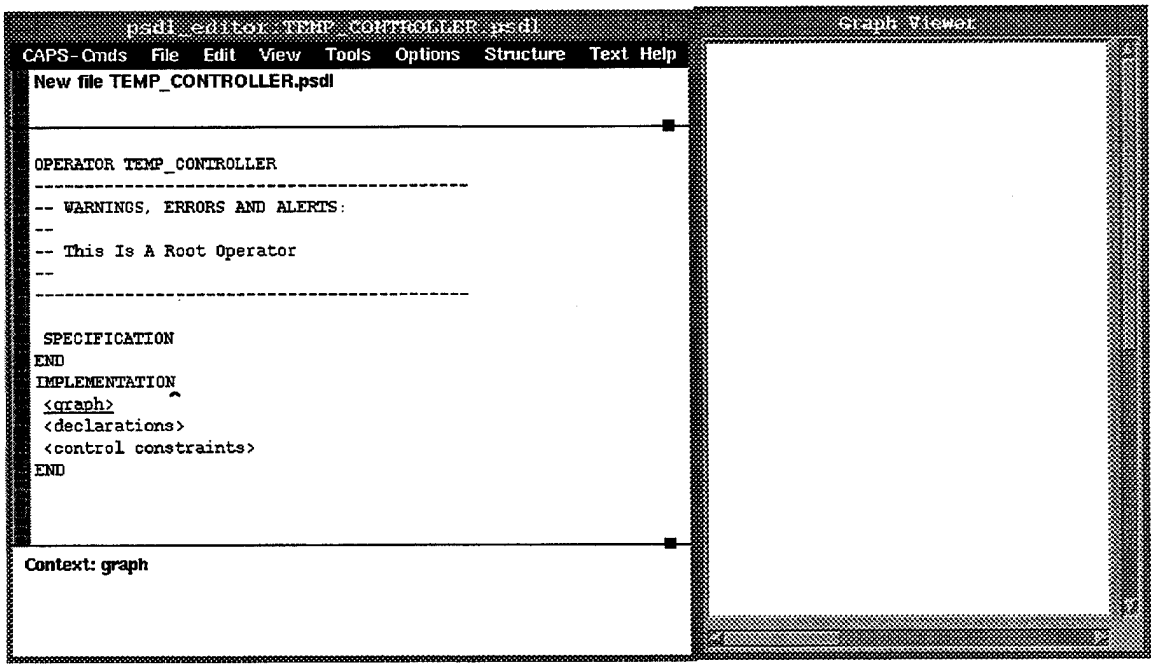


Figure 23. PSDL Syntax Directed Editor (SDE) and Graph Viewer.

Notice that the initial root operator in the SDE is the same as the name that you provided in the new prototype selection. TEMP_CONTROLLER is entered as a Stub in the SDE and acts as the operator for the total prototype. Also, the Graph Viewer is empty as we have not started to graph the prototype yet. That's next.

All of the pull-down menus in the SDE are active, however all of the necessary commands for PSDL editing and file saving are found in the "CAPS-Cmds" pull down menu. Do not use the "file" pull-down menus, because if you do SDE will not save a correct PSDL program.

From the SDE, invoke the CAPS Graphic Editor by using the "edit graph" command from the "CAPS-Cmds" pull-down menu. Figure 24 will appear.

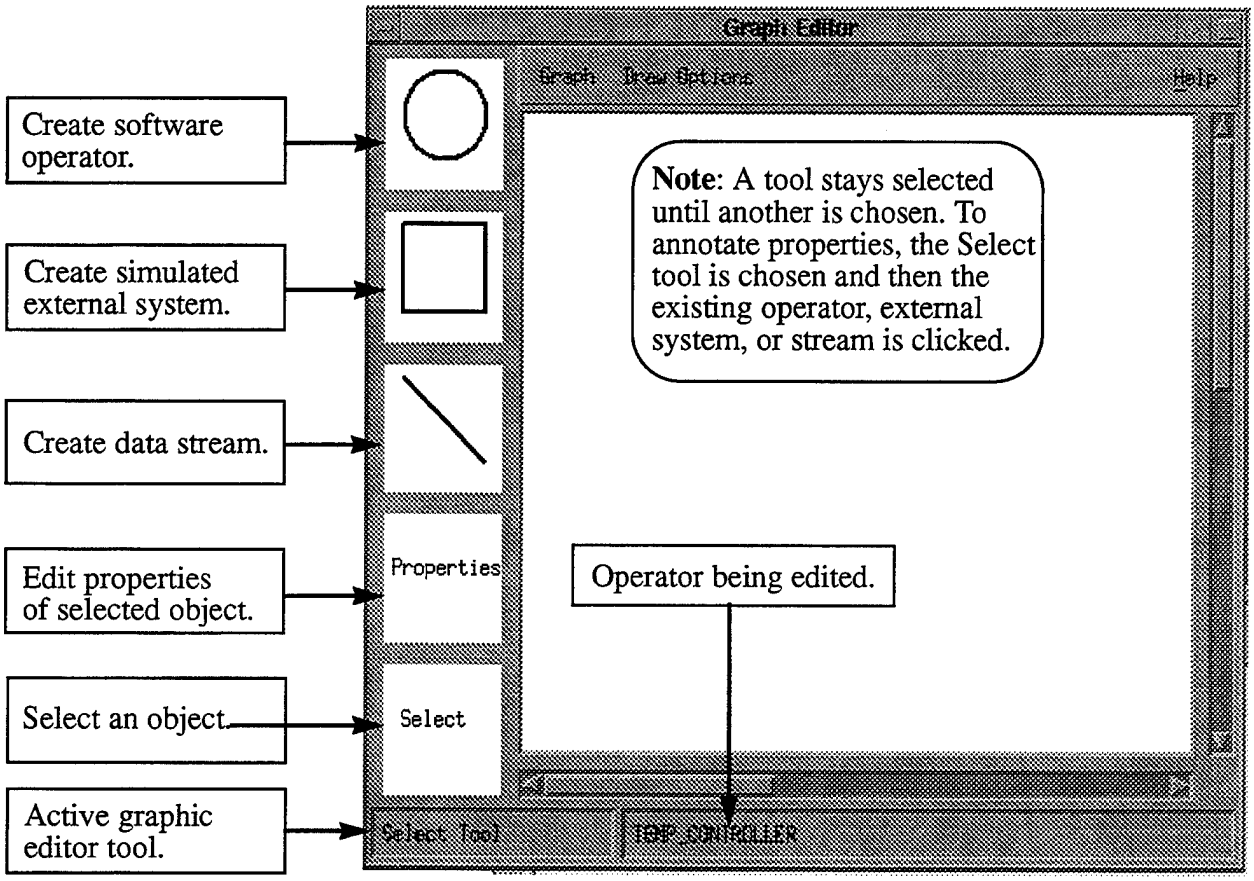


Figure 24. The CAPS Graphic Editor.

Once in the Graphic Editor, the designer draws the data flow diagram, enters operator names, inserts input and output streams, and enters some of the prototype timing information. Additional timing and control constraints are entered and implementation options are selected in the SDE. That will be covered after you complete the graph of the prototype.

The CAPS Graphic Editor is used first and foremost to lay out the data flow design of a prototype. Operators are linked together with data streams and given names. Context sensitive attributes are assigned to operators and data streams. These attributes are the maximum execution

time (MET) for operators and latency (LAT) for data streams. Recal that the MET is the maximum amount of CPU time the operator can use for execution and the latency of a data stream is a lower bound on the amount of time required for transmission of data along that stream. The figures and words that appear on the left hand side of the Graphic Editor (the Graphic Editor palette) are editing tools.

The functionality of the tools are summarized as follows:

CIRCLE.....Draw circular operators to represent proposed software components,

SQUARE...Draw rectangular operators to represent simulations of external systems,

LINE.....Draw data streams,

Properties...Assign properties to the selected operator or data stream,

Select.....Enable selection of an object in the graph.

The name of the active tool is displayed in the lower left portion of the Graphic Editor and the name of the operator being edited is displayed in the lower right portion.

Begin by putting in the software operators for the TEMP_CONTROLLER prototype. Select the Circle Tool, or software operator palette, and left click your mouse. Now move your mouse cursor into the working area and left click again. A circle will appear. Make six more by simply left clicking in six separate places so that your graph looks something like that of Figure 25.

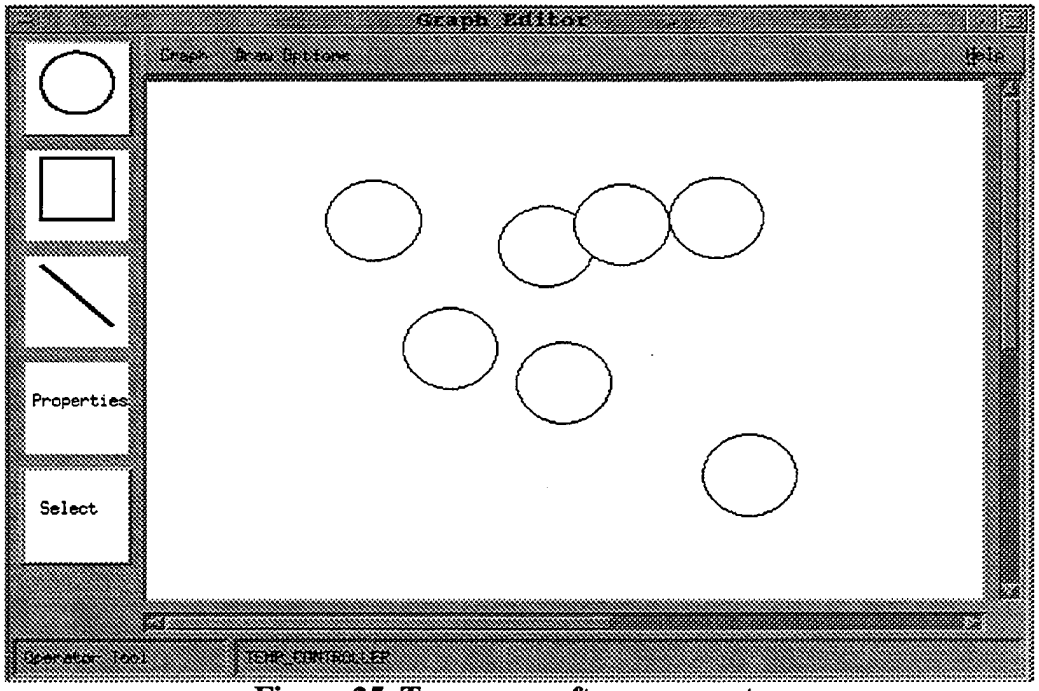


Figure 25. Too many software operators.

Whoops, you should only have four circles in your graph. You can delete some of them with the select tool! With your mouse, left click on the Select tool. Now move the mouse to a circle and left click again. The circle that you selected will be framed by square reference points. To delete the circle, press the delete key or backspace key on your keyboard. To select another object, simply move the mouse onto another circle and left click again.

You can also move and resize objects within your working space with the use of the select tool. Simply left click on the Select tool and then left click on the object you want to move or modify. Once you have left clicked on the select tool you can also make another object active by clicking on it without hitting the select button again. By depressing the left mouse button while on an object, and then moving the mouse, you can drag the object to where you want to place it. When you reach the point where you wish to place it, simply stop dragging the mouse and release the left mouse button. In the case of re-sizing, simply left click one of the reference points after selecting the object and move the mouse, letting go of the bottom when the object is as desired.

To label your software operators, click on the Select tool, select an object, and then click on the Properties tool. This will bring up the Properties_popup window seen in Figure 26.

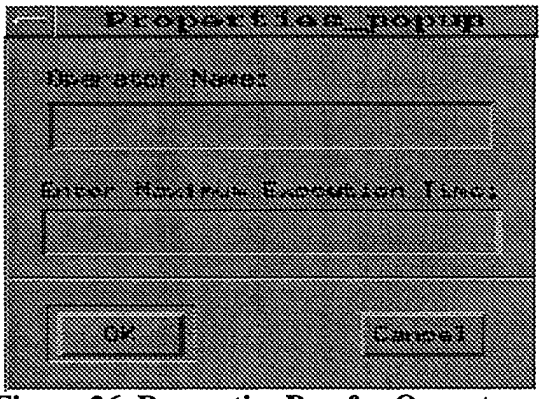


Figure 26. Properties Box for Operators.

Click in the Operator Name box and type its name. Then click in the next box or hit the tab key to enter the MET information, if desired. Now click "OK" or hit the <enter> or <Return> key. The units of milliseconds (ms) are automatically appended to the number entered in the Properties popup dialog box. If you enter units while in the Properties popup dialog box, your max execution times will not appear on the graph.

The TEMP_CONTROLLER prototype with all software operators labeled should appear similar to Figure 27 below.

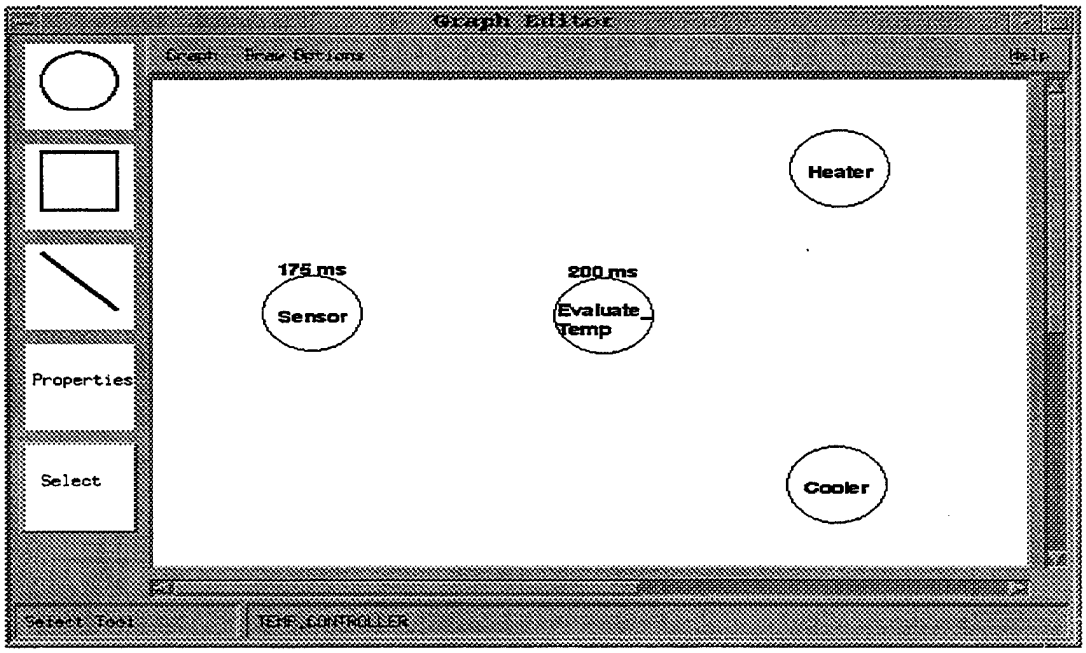


Figure 27. TEMP_CONTROLLER Prototype with Operators labeled.

Max execution time labels can be selected and moved like all other units. Be careful that you do not misplace them however, as the object they refer to should be obvious by their placement.

Next you will use the Line tool to enter data streams between objects. Move your mouse to the Line tool palette and left click. To start the stream, place the cursor in an object and left click. To end the stream, move the mouse to another object and left click again. Data Streams can originate or terminate in an object or outside of an object if they are external input or output. To start or end a stream outside of an object, double click the left mouse button. If curved lines are needed, create way points along the line by clicking the left mouse as you proceed to the termination point of the stream. When the line is completed it will curve to conform with the way points. Figure 28 shows the TEMP_CONTROLLER prototype with data streams entered.

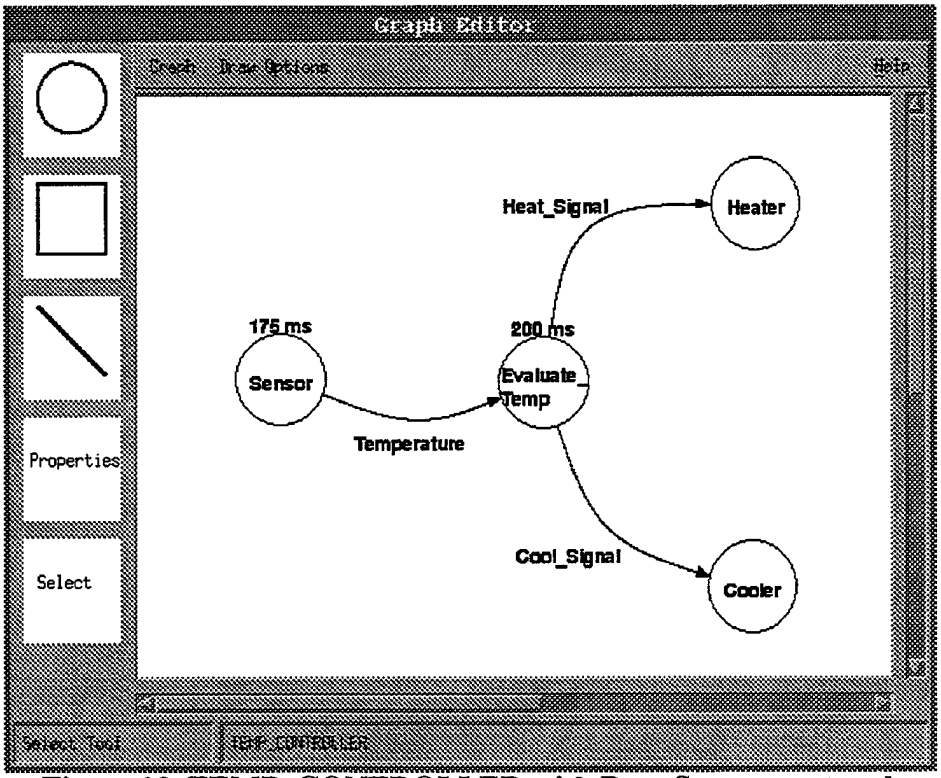


Figure 28. TEMP_CONTROLLER with Data Streams entered.

The data streams have properties similar to the other objects in that they have object names and a latency attribute. The Properties_popup for data streams is shown in Figure 29. Again, the latency attribute is measured in ms by default. The functional restrictions pertaining to

data entry for this window are the same as in the Properties_popup for operators. Note that the editor is smart enough to call up the right Properties_popup box.

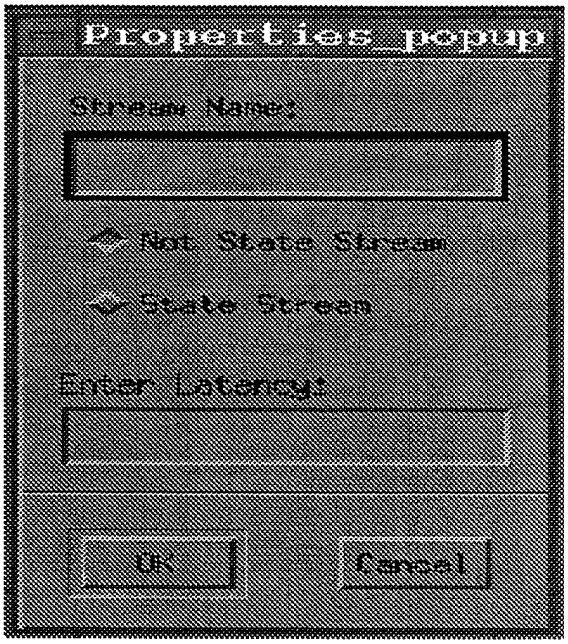


Figure 29. Properties Box for Data Streams.

The data stream popup box allows for indicating whether the data stream is a state stream or non-state stream. These are optional and have not been entered here. In the graphics editor, the default is non-state stream. In this edition of CAPS Release 1, the state streams are defined in the SDE. For more details on data streams, refer back to chapter two.

Data stream labels can be moved like other objects by using the Select tool and then dragging with the mouse. If a data stream name or latency label is deleted the entire data stream is deleted as well. If a stream must be modified (i.e. rename it or change its latency time), it should be done through the Properties_popup box.

The prototype data flow graph is complete and there is little left to do in the graphic model. We will now open up the SDE. To return to the SDE from the Graph Editor, drop the "Graph" menu on the menu bar and select "Return to SDE". The Graphic representation of your prototype will be saved automatically. With the possible exception of "Decompose", which will be discussed later, the other choices on both menus are self-explanatory.

The first thing you should notice when you return to the PSDL Editor (See Figure 30) is that stubs have been created (in alphabetical order) for all of the data streams that you created in

the Graph Editor. Stubs, in the form of Control Constraints, have also been entered for all Operators that you created. These too are in alphabetical order.

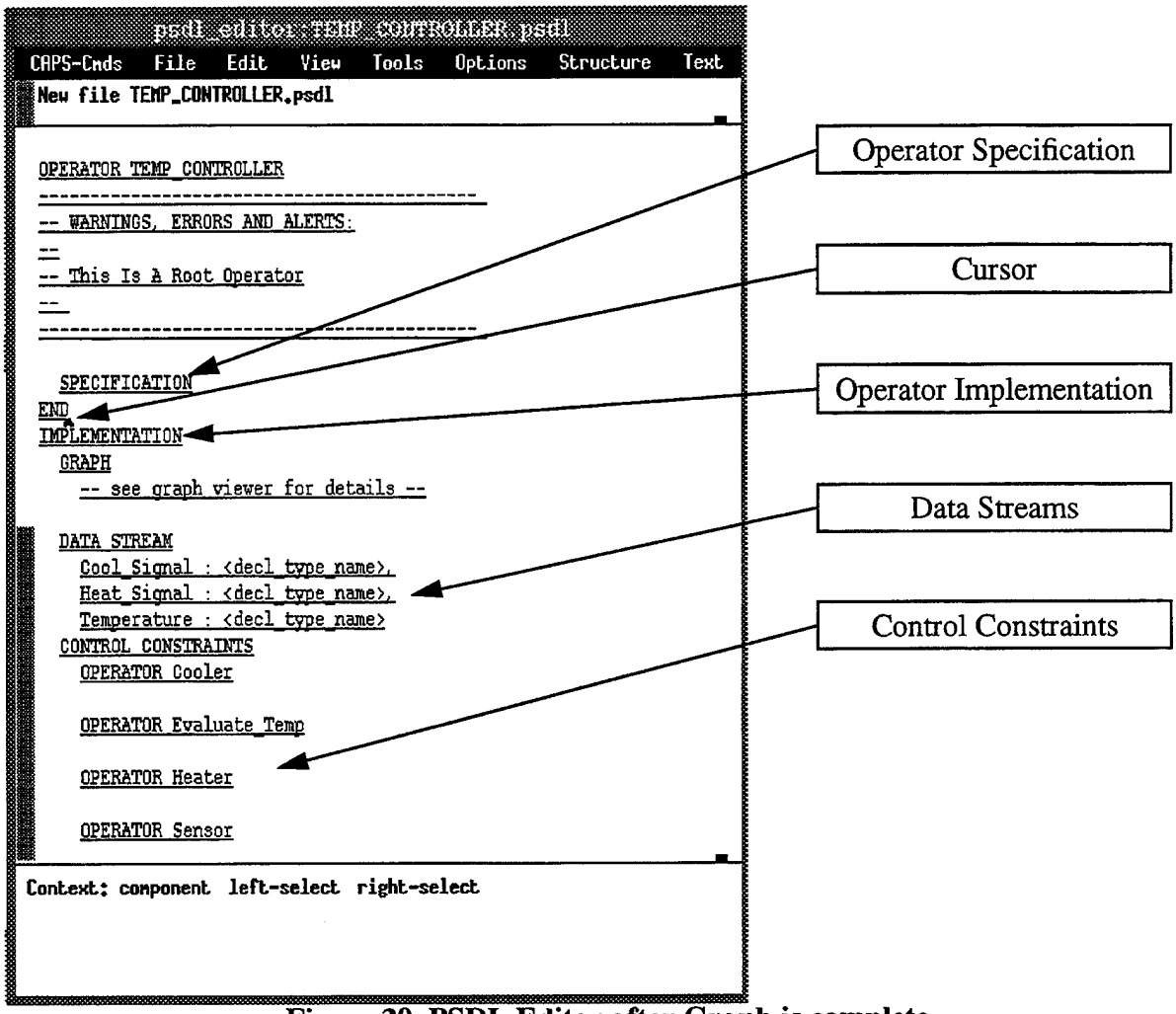


Figure 30. PSDL Editor after Graph is complete.

Once in the editor, you can use the mouse to move the cursor, which appears as a caret, or you can move it with the arrows on your keyboard. When you move the cursor to the end of "SPECIFICATION", the menu in Figure 31 appears at the bottom of the editor. Some of these selections are input via the Graph Editor, i.e., o_inputs_list, o_outputs_list and o_timing_info. You can enter these things in the SPECIFICATION, but if the information that you enter is inconsistent with that entered in the Graph Editor, the SDE will change it to make it consistent. Most important to remember is that the cursor is context sensitive and menus will automatically open based on where the cursor is placed. You will also know what part of the prototype is currently selected by the underlining that SDE inserts.

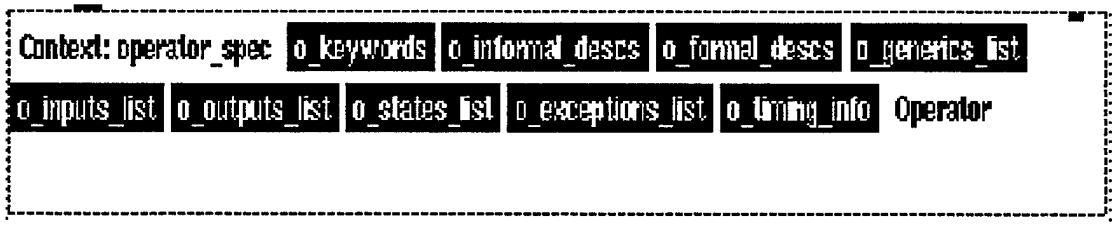


Figure 31. Specifications Tool Bar.

Since we are in the top level of the editor, the only things that we will enter into the SPECIFICATION are key words. Select "o_keywords", which stands for optional keywords; the "o" in all of these options stand for optional. The next display should look like that of Figure 32.

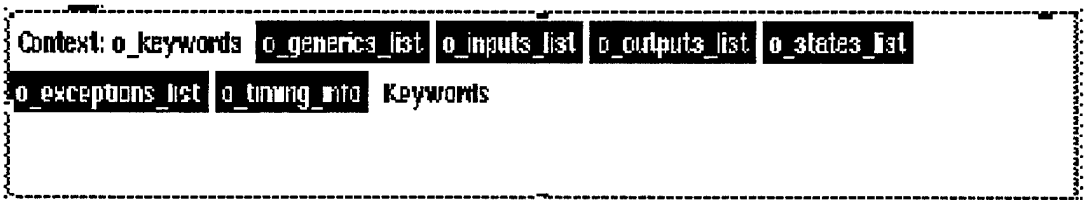


Figure 32. Keyword Tool Bar.

Note that the context of the cursor position now changes from operator_spec to o_keywords. When the Keywords Tool Bar opens, the SPECIFICATION section of your PSDL editor will change. A *place holder* will appear in the form of square [] or angel brackets <>. Any time you see square brackets, information placed inside of them is optional. The actual insertion under the SPECIFICATION line should be "[optional keywords]". Click on the Keywords option in the Keyword Tool Bar and the [] will be replaced by <identifier>. You are now ready to enter in

Keywords. Hit the return key after each keyword (commas will be inserted). Multi-word keywords can be separated by underscores. When you are finished you must hit return twice. This will save your entries.

If you make mistake, you can back space over incorrect text just entered or you can highlight (click of left mouse) that section of the prototype to be deleted (recall it will be underscored). Then hit <ctrl> <shift> <k> simultaneously. If you have an optional empty place holder you can eliminate it by moving the cursor up with the arrow key or by clicking elsewhere with the left mouse button.

Other items that you can add to the SPECIFICATION are informal descriptions and STATE DECLARATIONS. The informal descriptions will appear in { } and are similar to user/ programmer comments in other programming language. They are for human consumption only and are ignored by the PSDL compiler. The State Declarations are necessary to specify Data Streams that require initial values.

You can look at the graph at anytime by clicking in the PSDL where there is mention of the graph components. For example, positioning the cursor at the "OPERTOR TEMP_CONTROLLER" or below will bring up the graph at the top level (the only level at the current time). The Graph Viewer will come up automatically. If the graph viewer window is closed (or minimized), it will be labeled, "graph_edit". Do not be confused by this as it is still the graph viewer.

Now enter the declaration types for the data streams. Position the cursor before, in, or after the <decl_type_name> component of the DATA STREAM. A subtle requirement of the SDE should be mentioned here. When entering information applying to streams and control constraints of operators, you must enter them into the parent of the operator or stream. Left Click the place holder specified above and a menu bar of standard defined types will activate at the bottom of the PSDL Editor. Click on the type declaration "FLOAT" and that data stream will be defined for that type. Now do the other two. Actually, you would pick which ever type makes the most sense for your design. User defined types can also be accessed from this menu bar and typed in by the user but that is outside the scope of this guide.

More important however is the propagation of type declarations throughout the PSDL program when you define the data streams. Once you've selected the types for these three streams

within the parent operator, you merely left click anywhere else and the new declarations are propagated throughout the prototype as shown in Figure 33 below.

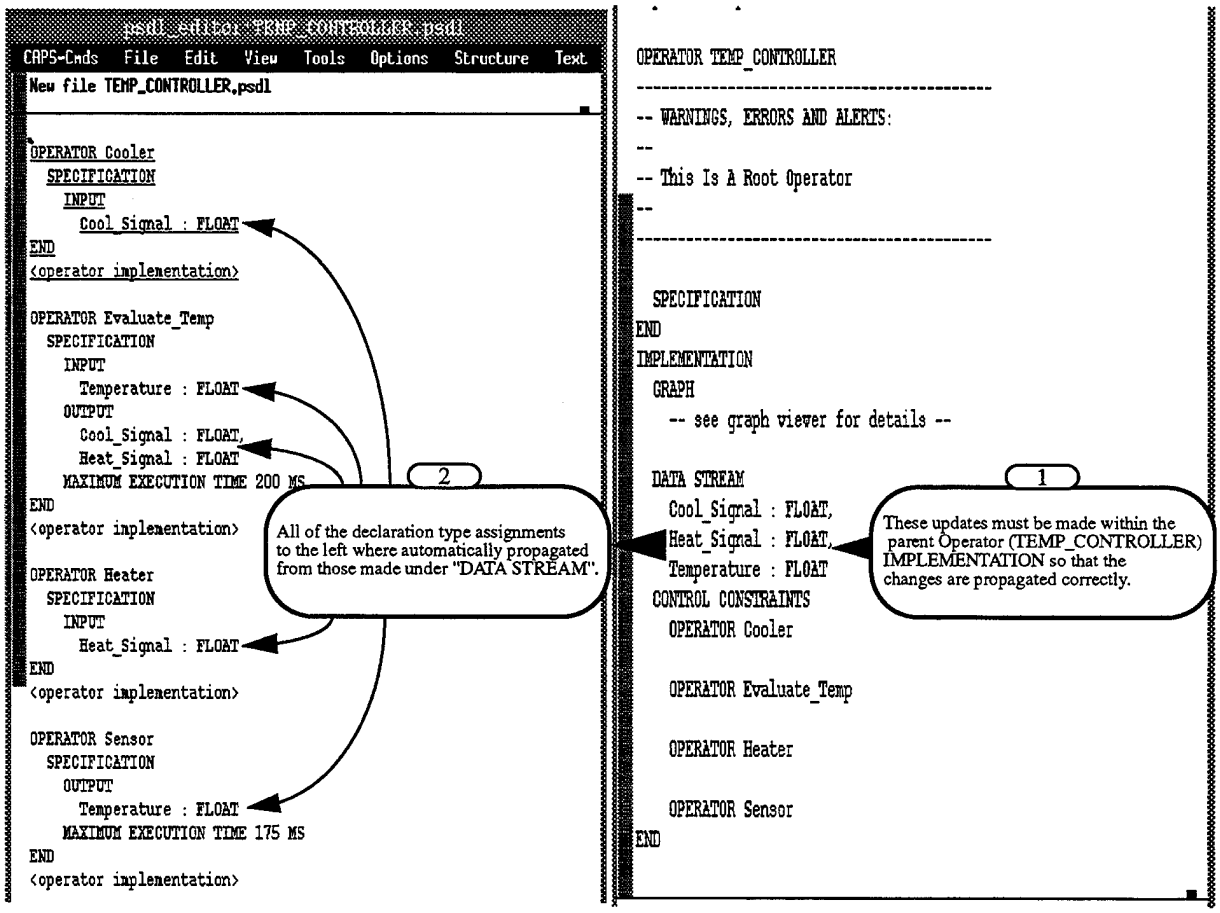


Figure 33. Complete PSDL Program showing Type Declaration propagation.

Also illustrated in Figure 33 is the PSDL Editor's ability to correctly assign data streams as either input or output within the Operator Specifications found in the top of PSDL program. The Maximum Execution Times that you entered in the Graphics Editor are displayed there as well.

Control Constraints can be modified directly from the PSDL Editor by using the mouse or arrow keys as previously mentioned. While in the graphics editor, two processes (Sensor and Evaluate_Temp) were deemed as time-critical and were given MET's. At this point, it must be decided whether they are periodic or sporadic operators. Suppose we decide that both should be periodic and that the constraints are as follows: $PER_{Sensor} = 190ms$, $PER_{Evaluate\_Temp} = 210ms$, $FW_{Sensor} = 185ms$, and finally $FW_{Evaluate\_Temp} = 205ms$.

In order to accomplish this, we first left click just past the control constraint we wish to edit; "Sensor". Then the menu bar at the bottom of SDE changes to that shown in Figure 34. You click on "optional_period", and then the menu in Figure 35 appears. Click "Optional_Period" and the operator "Sensor" has its first constraint under it. You do the same procedure for entering the time (not shown here); i.e. left click "Time_Expression", enter an integer for the placeholder (should be 190), hit return, select the desired units from the menu bar, and finally click where you want to go next.

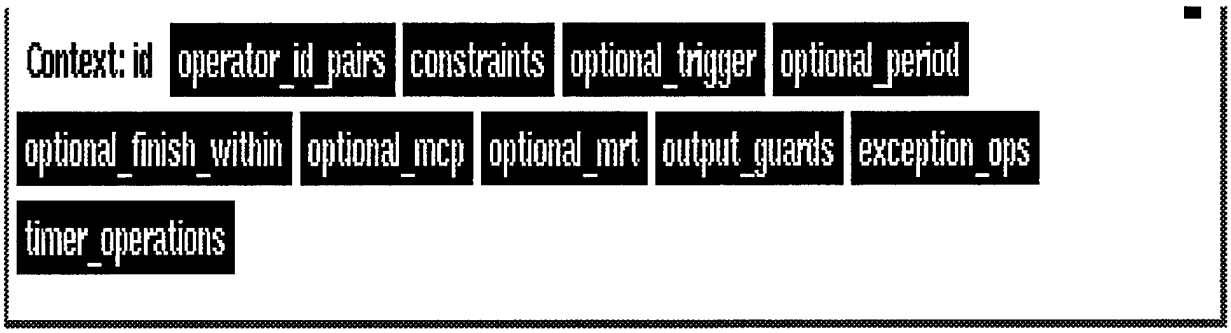


Figure 34. Control Constraints Menu Bar.

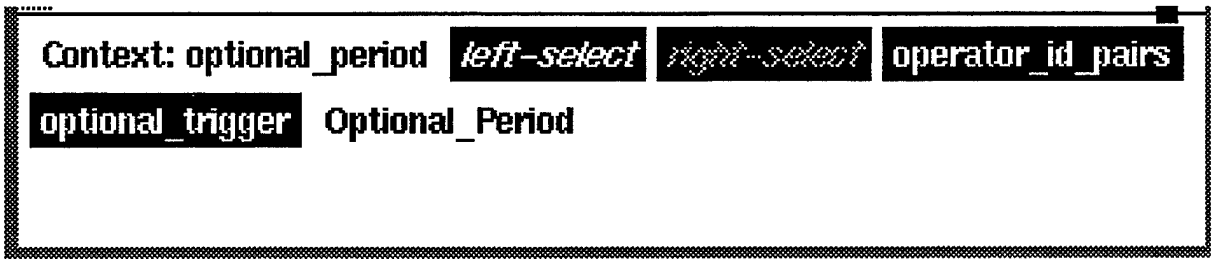


Figure 35. Optional Period Menu Bar.

After you enter in the constraints for the two time-critical operators, your PSDL should look like that pictured in Figure 36.

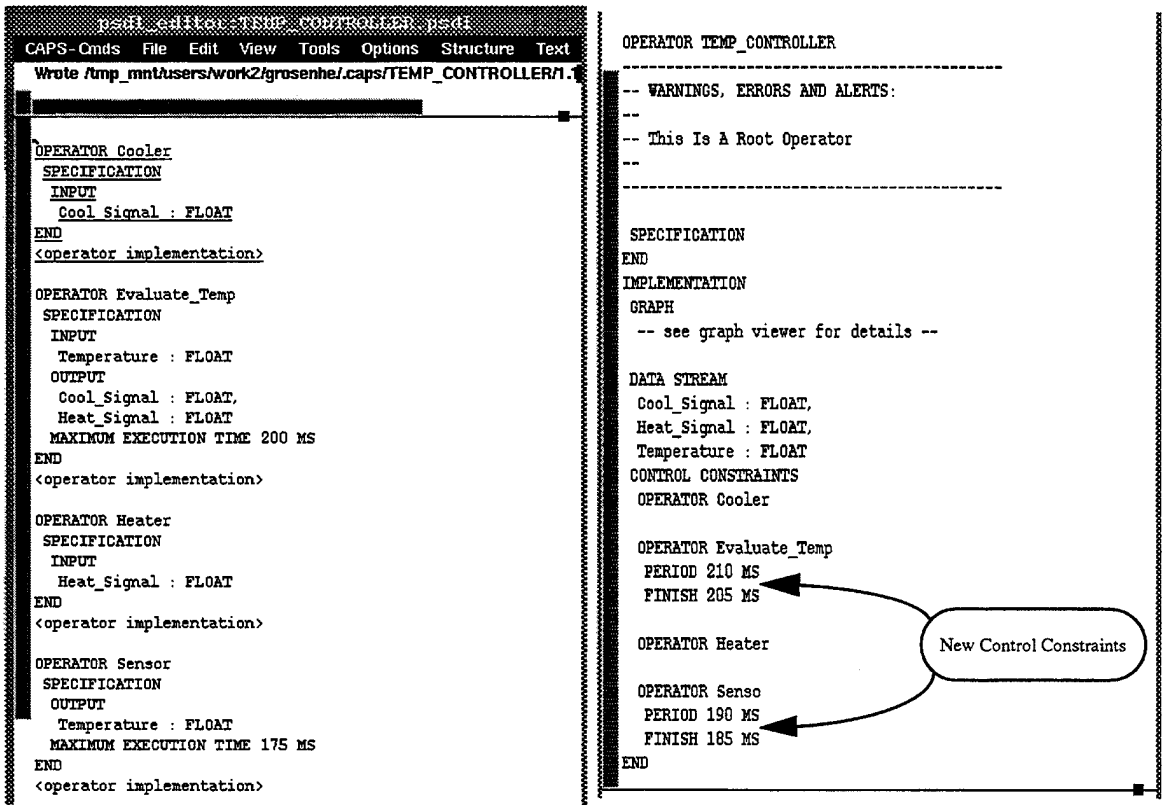


Figure 36. Complete PSDL Program with Control Constraints.

But what if we wanted to decompose one of our operators? Recall the CAPS prototyping process from chapter two. When you have a component, you first look to see if it can be implemented with a component from the software base. If not, it must be determined whether the operator should be decomposed, If yes, you will go through the same thought process just described to specify the sub-components of the operator. When dealing with operators that should not be decomposed, they must be implemented by hand (currently requires Ada implementation). So then, the question here is how to decompose an operator.

First select "edit-graph" from the "CAPS-Cmds" in the SDE menu bar. You should get a window that looks like Figure 28 again.

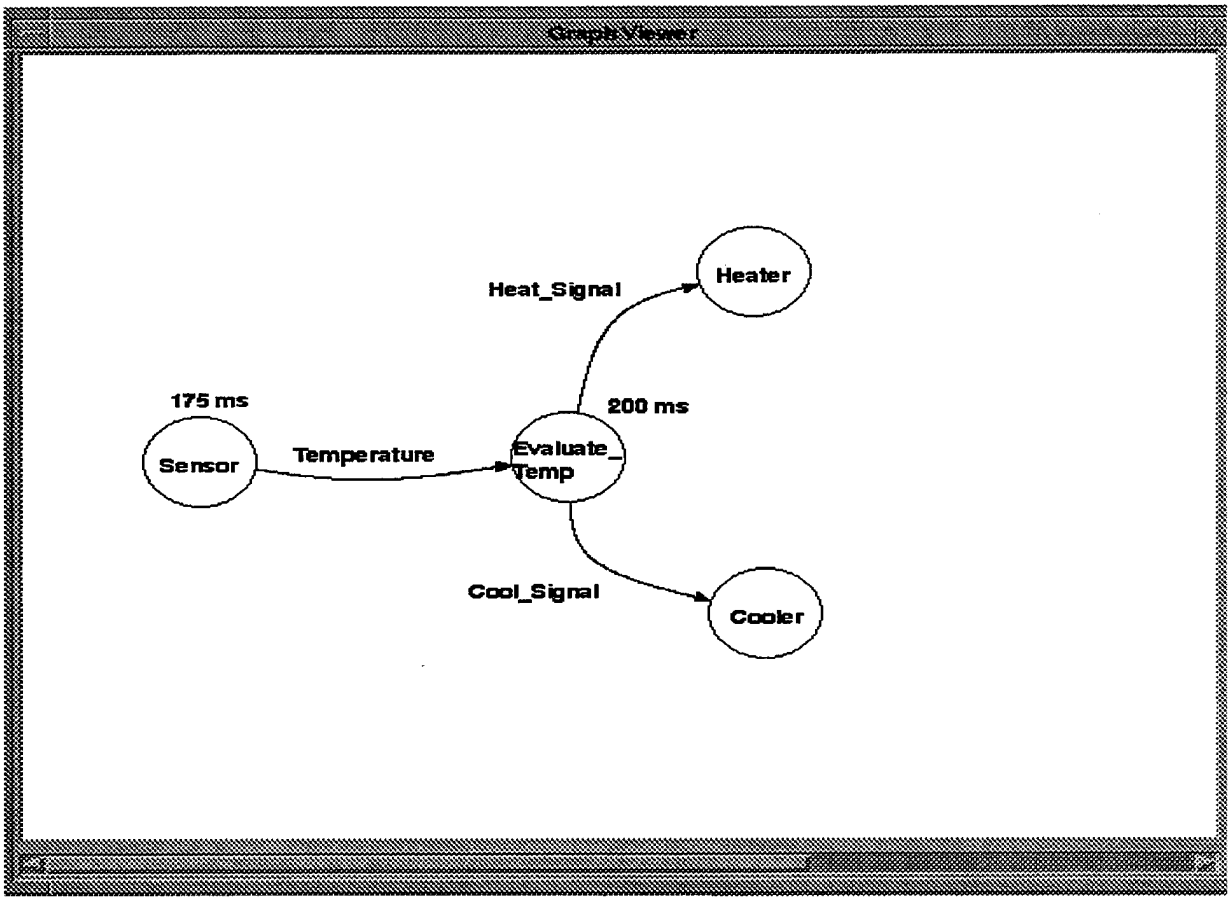


Figure 28. (Repeated).

Select the Sensor Operator and then select "Decompose" from the "Graph" menu bar. It will automatically open up a Graph Editor that looks like Figure 37 below. The Graph Editor shows the data stream, "Temperature", as an external output from the Sensor operator. This is information that you need to keep in mind as you decompose the Sensor. It must be maintained for consistency and is brought forward so that the user does not have to look back at the parent operator.

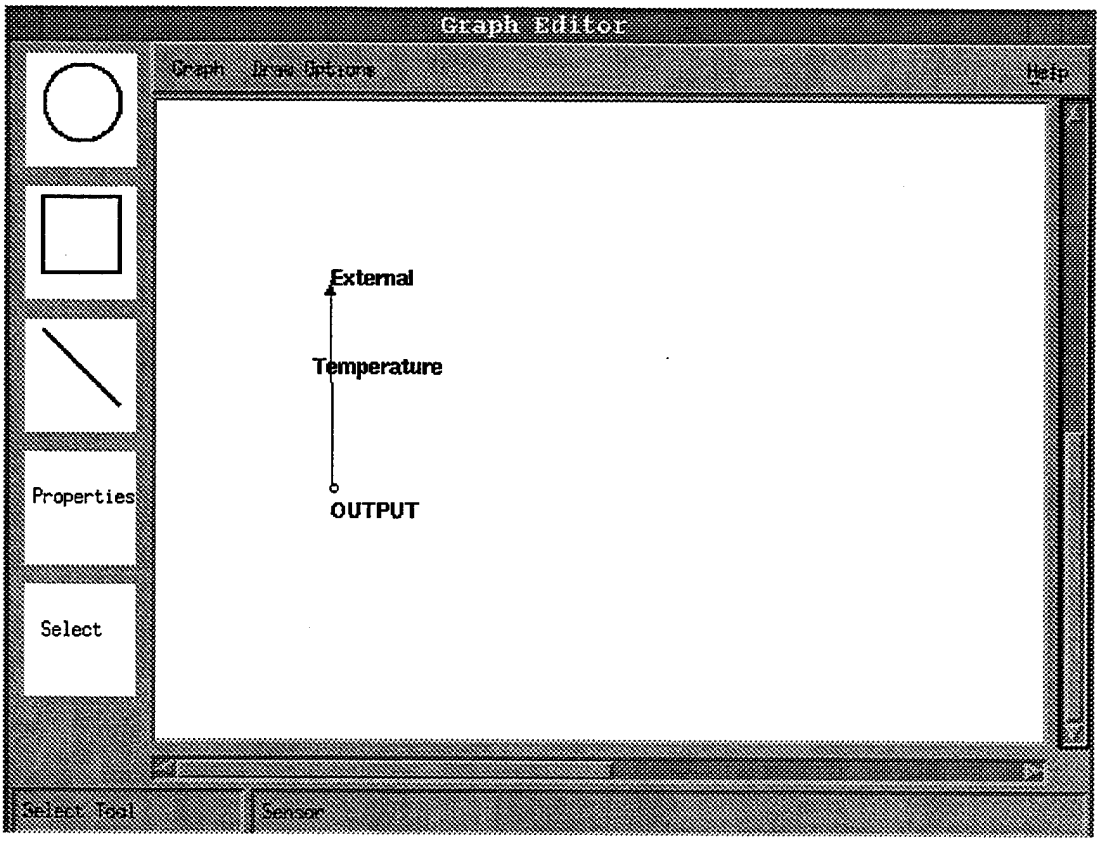


Figure 37. Operator Sensor after initially selecting decompose.

To decompose Sensor, you draw the appropriate data flow diagram just as before. This diagram, however, is the internal workings of the composite operator "Sensor". An example is given below in Figure 38.

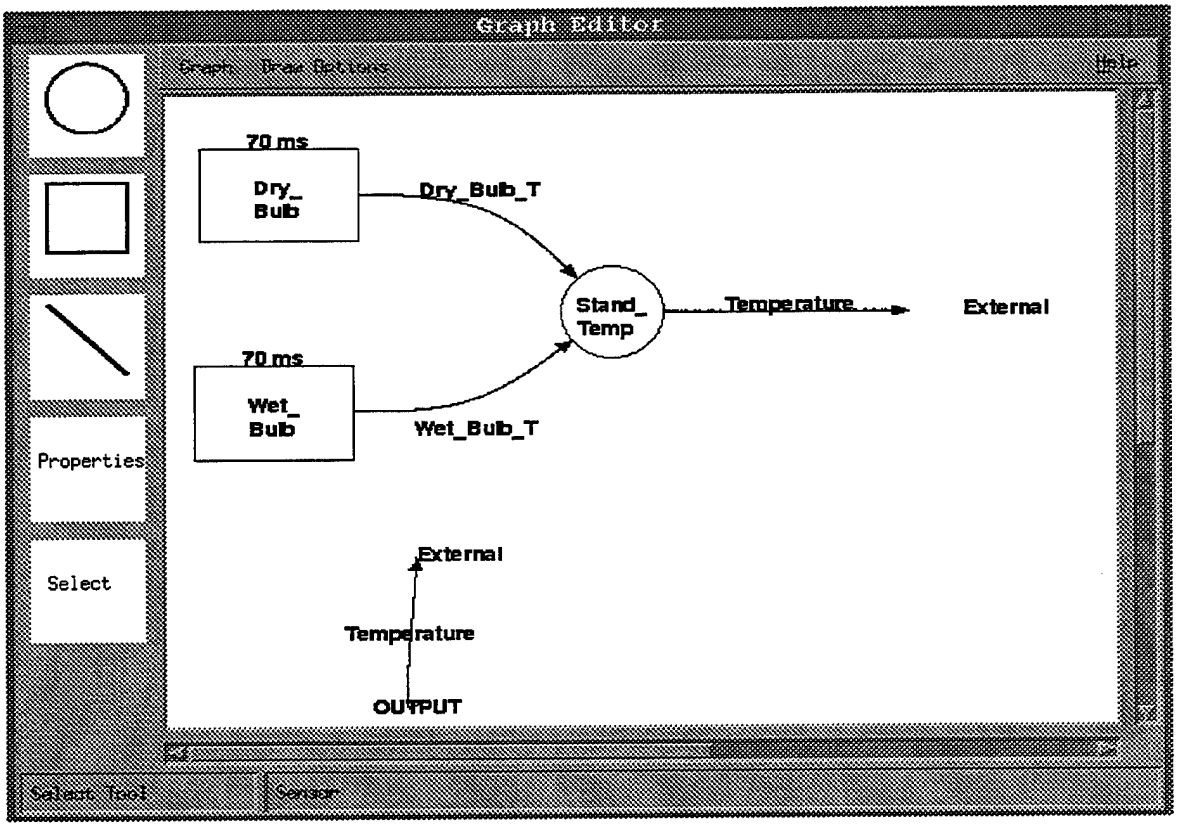


Figure 38. Operator Sensor decomposed.

Now that you are done with the editor, you can either go back to the previous level data-flow graph or return directly to SDE (choose to go back up this time). Those options are both under the "Graph" pull-down menu as "Edit Parent" and "Return to SDE" respectively. If you go back to the top level diagram, you should notice that the "Sensor" operator is different as shown in Figure 39. This doubled circle denotes that the operator is composite and has been decomposed.

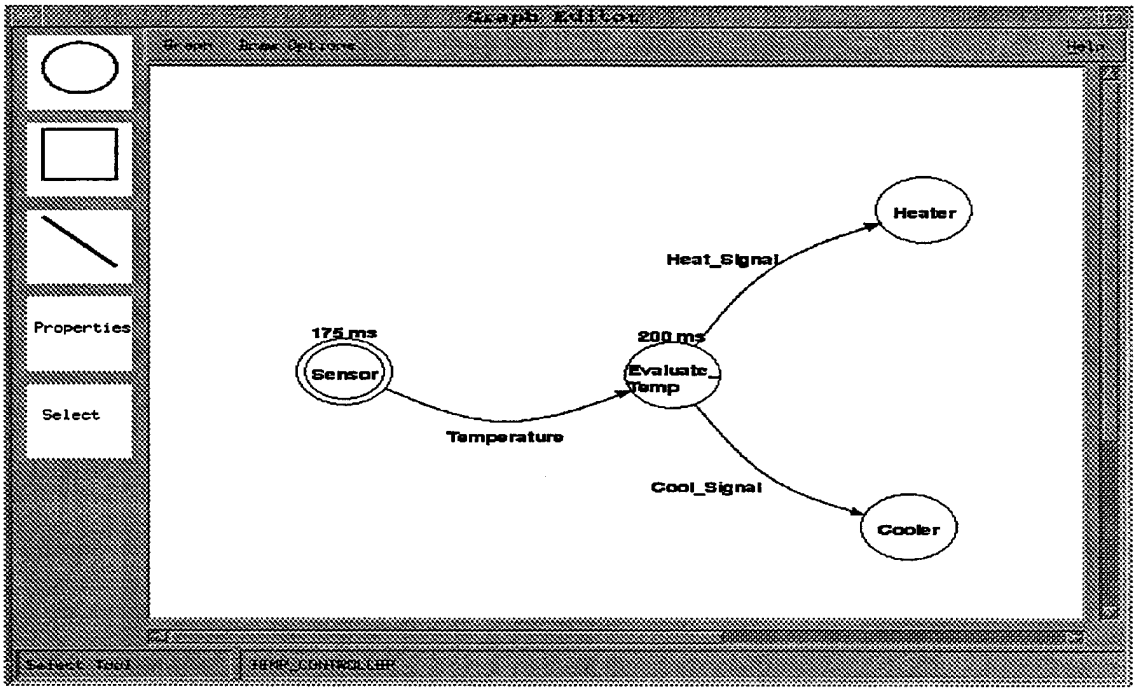


Figure 39. TEMP_CONTROLLER Prototype with Sensor marked as decomposed.

The PSDL file in the SDE will reflect the changes made and will add new operators, external simulators and data streams as necessary. The declarations and control constraints for these objects are modified as previously described in this guide.

You should now be able to build a prototype in the graphic editor, make type declarations, and define control constraints in the SDE. The following table provides a quick reference on which type of information can be entered into each editor.

Graphic Editor Information	Syntax Directed Editor Information
vertices & edges (operators & data streams) operator names data stream names operator maximum execution time data stream latency time operator color & shape	control constraints data stream types timer declarations state declaration & initialization user defined types operator & type implementation selection

Table 2. Summary of editor inputs.

If you look back at the complete PSDL program in Figure 36, you will notice one more set

of placeholders that haven't been filled in. These are for the "operator implementation". As discussed earlier, for each component in your design, you must either find a component within the software base that can fill its needs, further decompose it, or implement it in Ada. For instructional purposes here, simply go to each placeholder, click on it, and select "Ada_Implementation". Obviously, if this were a real application, someone would eventually have to implement those atomic operators. Once the prototype is complete, save your work in the PSDL Editor by "PSDL-Save" or "PSDL-Save-Exit" from the "CAPS-Cmds" button on the menu bar. The next step would be to Translate the prototype to generate the code that will tie everything together. After that, you would invoke the Scheduler which determines a schedule for the system. And finally, it would be compiled to create executable code. Don't go any farther as there are problems, besides the fact that you don't have implementations for your operators, that will be covered in section C.

2. SDE Menu Functions

Because the last section was designed to be a simple example, the menu of the SDE was mostly excluded with the exception of some of the options under "CAPS-Cmds". Recall that the "File" menu should not be used. This section was added because there are several options that the designer should be familiar with. The others can be experimented with to get some exposure as free time dictates.

Under the "Edit" pulldown menu are several options that most users are already accustomed to. The cut, copy, paste, and delete are in most editors today. The SDE is no exception. This menu gives you a choice of either text or a structure that you've already highlighted, and acts like any other editor with respect to use. Getting accustomed to these can enhance the designers productivity as they speed development time.

The "View" pulldown menu deals with what representation of the abstract syntax tree you wish to see. You can select any number of different views or a combination of them. While this is fun to experiment with, the default view is the best for the beginning designer.

The "Tools" pulldown menu is also for a more advanced user. A designer that is familiar with the concepts of the sub-menu items below this option will quickly grasp their intent as they are very common in the Unix toolset.

The "Structure" pulldown menu deals with moving around in the edited prototype. Most of this is also accomplished more easily with the tab, space bar, and especially the mouse. The

one option that must be mentioned, however, is the "ascend-to-parent". Sometimes the user can get confused as to what part of the abstract syntax tree is being displayed in the editor and there are times when the structure highlighted must be walked back up the tree. For example, when a designer decides to quit work on a prototype before assigning types to the streams. Before quitting, the prototype would look something like that displayed in Figure 40 below. Notice that the placeholder says <decl_type_name>.

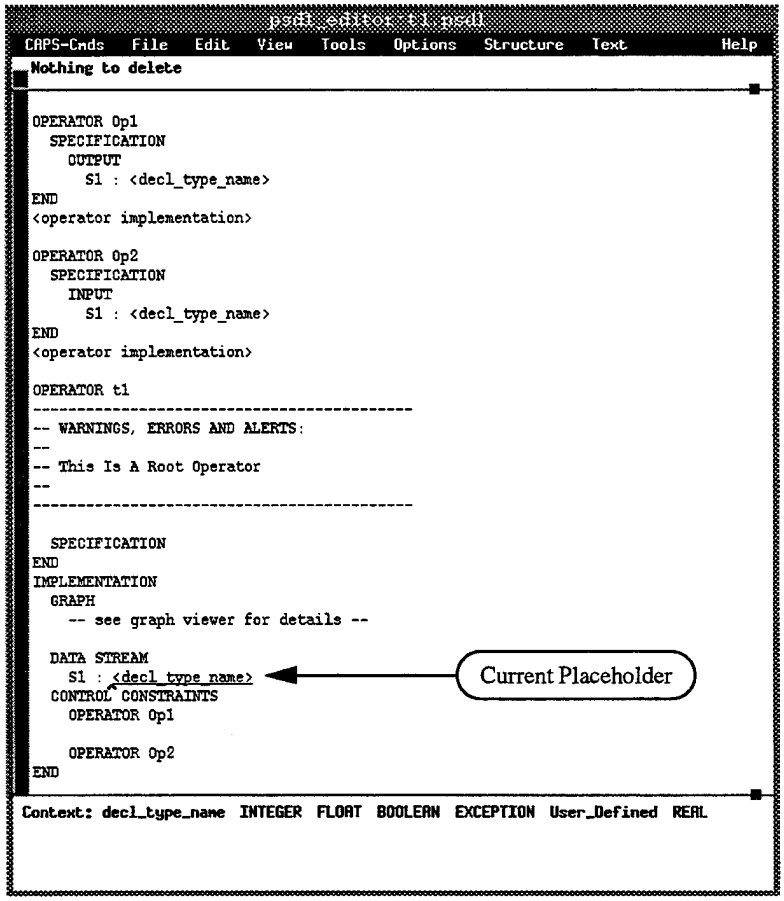


Figure 40. Data Stream without Type Declaration.

If the designer were to exit the SDE, saving the prototype without fully defining the stream S1, the placeholder would be different upon re-editing the prototype. Because the SDE checks to ensure that any prototype being loaded into the editor is syntactically correct, the stream must be defined. To keep from getting an error then, SDE puts an identifier in the placeholder. Now when you open the prototype, you see what is displayed in Figure 41.

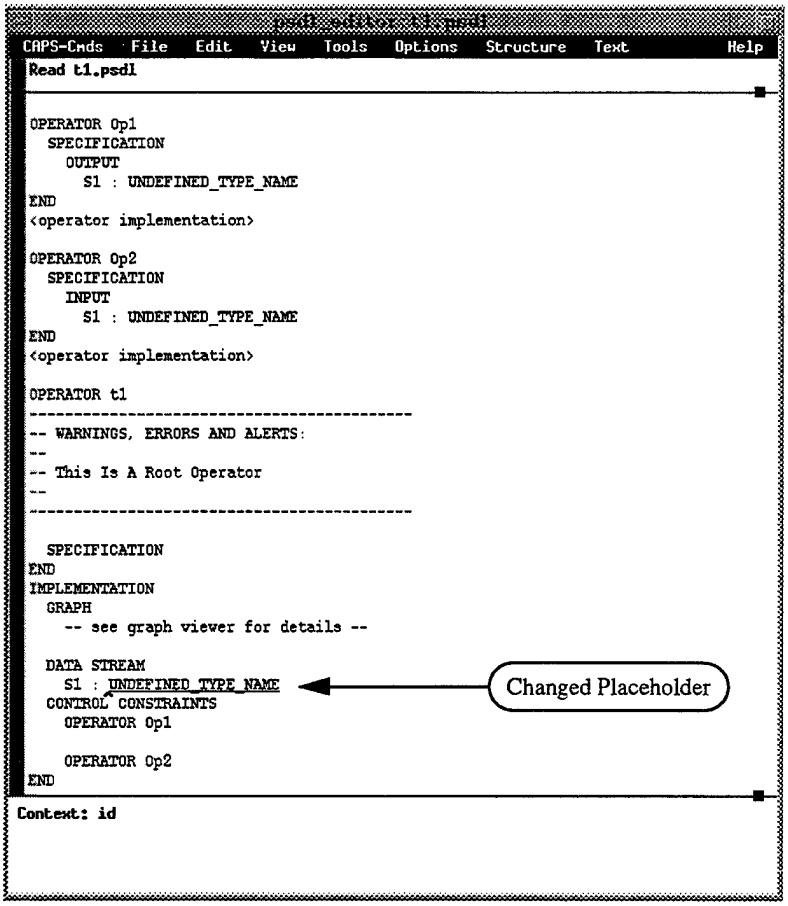


Figure 41. Data Stream with changed Type Declaration.

The designer may now want to make the stream an integer. Instead of indicating a type_decl_name as before, it shows the placeholder as an id. The designer must select the id, UNDEFINED_TYPE_NAME, and delete it using the "delete-structure" option in the "Edit" menu. This will make the placeholder appear as <identifier>. The next step is to walk up the syntax tree. This is done by selecting "ascend-to-parent" in the "Structure" menu. After doing that, you should notice that the context (shown in the Help Pane at the bottom of the SDE) now shows the placeholder to be a decl_type_name, which is desired. Delete the placeholder one more time and you are back to what was there before you quit. In addition, the Help Pane now shows all the available transformations for a decl_type_name.

Alternately, one can first select the id, UNDEFINED_TYPE_NAME, and then "ascend-to-parent" in the structure menu, followed by selecting the "delete-structure" option under "Edit".

The "Text" pulldown menu provides more ways of moving around, provides for search

and replace techniques, and has an undo option. And finally the help is the last pulldown menu but is very limited in its current implementation.

Finally, some of the useful keystrokes of the editor are listed below. Recall that the delete keystroke was covered earlier in the example.

Control-Shift-K.....Delete Structure,
Control-Shift-P.....Move up one level in the parse tree,
Control-h.....Delete previous character,
Control-v.....Advance one page,
Control-d.....Delete next character,
arrow keys.....move up, down, left, right,
delete or backspace.....Delete previous character.

There is no "undo" command in the Syntax Directed Editor, so the user must be careful. If a major problem occurs during editing, the best option is to select "exit" from the "CAPS-Cmds" pulldown menu and exit without saving.

B. NEW FUNCTIONALITY

1. Scheduling Constraints

a. General

All time-critical operators are non-preemptable under the current CAPS model, meaning that once they start, they will run until done. And as previously mentioned, these operators are statically scheduled; before runtime. The other operators, however, are preemtable, and are scheduled dynamically at runtime. This means the hardest part of ensuring that a real-time system will run correctly is ensuring those time-critical operators can be scheduled. Non time-critical operators simply use whatever time is left during runtime, including the time left over when an operator finishes early. For that reason, most of the concentration here will be on time-critical operators.

Recall that when an operator has a MET (maximum execution time), it is time-critical and is classified as periodic or sporadic. The previous definition of MET, the maximum amount of CPU time the operator can use for execution, is right on for atomic operators. In the case of composite operators, however, MET is defined a little more loosely as the maximum time needed along any thread of control within the operator itself. In addition to the above definition, it must be realized that the MET encompasses data triggering checks, stream reads, execution guard checks, execution of the operator, output guard checks, stream writes, and exception handling [Ref. 11]. Table 3 recaps the two classifications of a time-critical operator.

Periodic Operators	Sporadic Operators
Maximum Execution Time (MET) Period (PER) Finish Within (FW)	Maximum Execution Time (MET) Minimum Calling Period (MCP) Maximum Response Time (MRT)

Table 3. Classifications of Real-Time Operators.

As shown above, and previously discussed in chapter two, periodic operators also have a PER (period), and a FW (finish within) time where the PER depicts the frequency in which the Scheduler makes a processor available for execution and the FW denotes how long from the start of the PER before execution must be completed. Figure 42 below is provided to help make this concept a little clearer. The figure shows that periodic operators are triggered by temporal events occurring at regular intervals; thus the PERiod, which is the span of time between two successive activations.

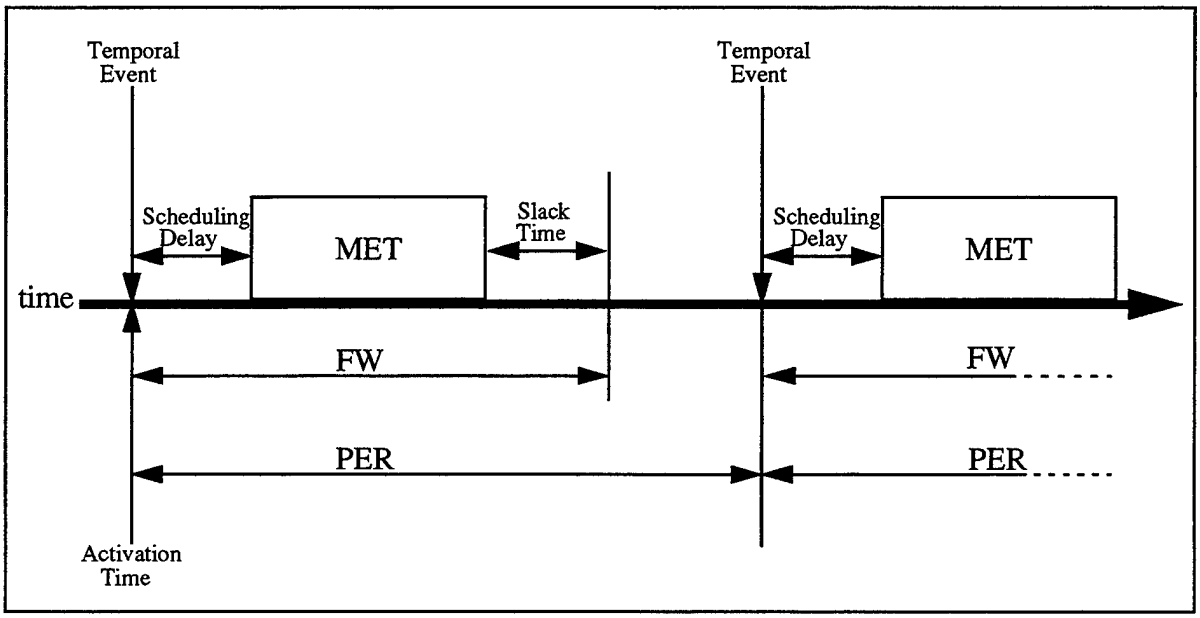


Figure 42. Timing Constraints for the Periodic Operator.

In addition to the MET, a time constrained operator that is sporadic has a MRT (maximum response time) and a MCP (minimum calling period), where MRT refers to the maximum amount of time between the arrival of new data on the input data stream(s) (which triggers the operator) and the time when the last output is put on the output data streams, and

MCP refers to the lower bound on the delay between two subsequent arrivals of triggering data on the input. MCP is actually more of a constraint on the operator(s) that produce a triggering data stream. Figure 43 shows sporadic operator constraints pictorially.

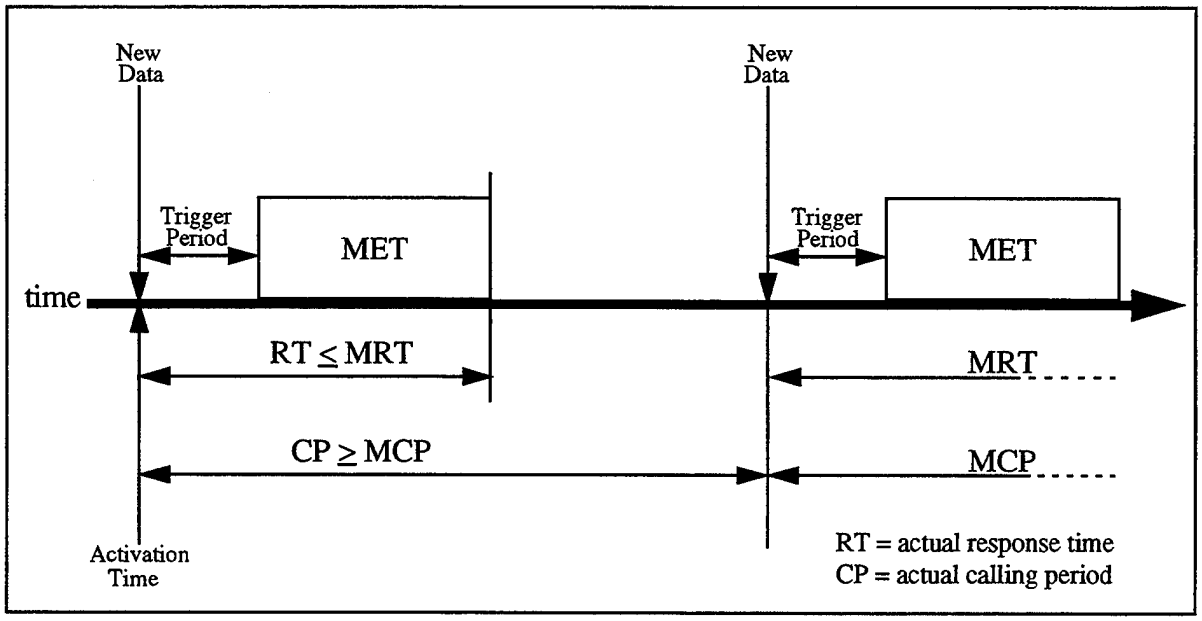


Figure 43. Timing Constraints for the Sporadic Operator.

There are two other timing constraints provided by PSDL that haven't been mentioned yet. They are for dealing with distributed systems, and because they are important, are covered here. However, these are not used in the current enhancements to the SDE and therefore will not be mentioned after this paragraph. The first is *Latency* (LAT), which denotes the maximum delay that can occur between a producer operator writing to a stream and a consumer operator reading that same stream. This can become a problem when dealing with networks and therefore must be accounted for in such cases. The second constraint for distributed systems is the *Minimum Output Period* (MOP), which is the amount of time an operator must wait before issuing another write to an output stream. Both of these constraints are due to the possible delays introduced by network traffic, and restrict an operator from reading/writing from/to a stream before the appropriate LAT/MOP has elapsed.

b. Data Triggering Semantic Checking

The specific constraints that must hold in order that a prototype is schedulable are many and complex. Because of this, only the simpler ones that are directly involved in changes

to SDE are mentioned. Also, because the focus of this thesis is concerned with making SDE more productive by adding schedulability checking, the theories behind schedulability rules will be discussed lightly and not proved. For a much more in-depth look at the fundamental theory of the scheduling of distributed hard real-time systems, see reference [Ref. 11, 16].

After gaining some familiarity with the concept of periodic and sporadic operators, several relatively simple, but important, constraints that must be imposed on the producer and consumer operator connected via one or more streams that need to be adhered to become apparent; constraints that can be checked while in the SDE.

If the producer of a stream has a period that is shorter than that of the consumer of the same stream, there is a strong likelihood that a problem will develop whereby the consumer can not stay caught up. For that reason, the rule is that the Consumer's PER must be less than or equal to that of the Producer if they are connected by a dataflow stream.

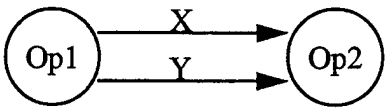
Sporadic operators are not regularly firing operators. They fire based on the arrival of new data. However, if a sporadic operator has no data trigger, it will be defaulted to a periodic operator with period equal to MCP to meet the need for some sort of conversion for periodicity and thus schedulability.

The next constraint involves a time-critical producer with a non time-critical consumer connected by a dataflow stream. There is a strong likelihood that the high priority producer will overflow the dataflow stream because the consumer can fire only when free time is available. Having the producer put data on the dataflow stream regularly with the consumer using the data when time is available simply will not work.

The last two constraints to be checked by the SDE involve the situation where a stream is produced by a non time-critical operator and consumed by a time-critical operator. If the consumer is triggered with BY ALL, the designer should be warned that possible overflow could result because if the producer gets enough time, it could produce new data for the stream before the next execution of the consumer has commenced. Finally, if the consumer is triggered with BY SOME, the designer should be warned that possible data loss could result because, just as before, the producer could get enough free time that it writes to the stream before the consumer reads the old data; the difference being that this time the old data would just go away.

Table 4 below shows the semantic data trigger related checking that is accomplished by the SDE. As alluded to earlier, cases not discussed nor shown below are either

trivial ones that call for no conditional checking or those which require sporadic operators to be converted. As SDE does not do any converting of sporadic operators for semantic checking, these can not be checked. Those cases can be found, however, in reference [Ref. 11].



P = Periodic Operator
S = Sporadic Operator
NTC = Non Time-Critical

Op1	Op2	Data Trigger	Remarks
P	P	By All	If $PER_{Op1} < PER_{Op2}$ then Error:Producer PER can't be less than that of Consumer
P	P	By Some	If $PER_{Op1} < PER_{Op2}$ then Warning:Possible Data Loss
P	P	None	If $PER_{Op1} < PER_{Op2}$ then Warning:Possible Data Loss
P	S	None	Warning: Sporadic Consumer Operator will have MCP as default period
S	S	None	Warning: Sporadic Operators will have MCP as default periods
P	NTC	By All	Error: Possible Stream Overflow
P	NTC	By Some	Warning: Possible Data Loss
S	NTC	By All	Error: Possible Stream Overflow
S	NTC	By Some	Warning: Possible Data Loss
NTC	P	By All	Error: Possible Stream Overflow
NTC	P	By Some	Warning: Possible Data Loss
NTC	S	By All	Error: Possible Stream Overflow
NTC	S	By Some	Warning: Possible Data Loss
NTC	S	None	Warning: Sporadic Consumer Operator will have MCP as default period

Table 4. Semantic Data Triggering Constraints to be done by SDE.

c. Timing Constraint Semantic Checking

The timing constraints attached to a time-critical operator are subject to multiple restrictions to ensure the possibility that the prototype is schedulable. As already mentioned, there are more than is described here; cases which fall in either the trivial or too hard category. Below is the first of several restriction that must hold.

$$\mathbf{MET \leq FW \leq PER}$$

If the MET was allowed to exceed the FW, the operator could use more CPU time than its deadline allows; this clearly can not be allowed. The second inequality must hold for a single CPU schedule because if the operator was allowed to finish after the period was over, it would delay the start of that operator's next activation.

The Maximum Execution Time Theorem [Ref. 11] states the following must hold for the set of all periodic operators in order that the prototype be schedulable on a uniprocessor.

$$\mathbf{MAX\ (MET) \leq MIN(PER)}$$

This in no way guarantees that the prototype is schedulable; we only know that it isn't if this is violated. There are two possibilities here.

In the case where the same operator had constraints such that MET > PER, it would not be schedulable even with a multiprocessor; that case is covered above.

In the second case, where the MET and PER belong to different operators, the constraint becomes apparent when you think about the fact that when violated and dealing with only one processor, the operator with the smaller PER will be blocked longer than the span of its PER while the operator (remember these are uninterruptable) with the larger MET executes. If the second operator is blocked longer than its PER, it can't possibly make the deadline. So then it makes sense that the smallest PER must be greater than the largest MET.

The first question that comes to mind is, "How many processors will do the job?". This question is answered for periodic operators with the Load Factor, which is cited in many sources dealing with scheduling. It shows the minimum number of processors that will be required to schedule the prototype.

$$\left\lceil \sum_{x} (\mathbf{MET_x/PER_x}) \right\rceil$$

Again, this does not guarantee schedulability. It only guarantees that without the number of processors specified, the prototype is not schedulable. Intuitively, this formula states that the larger the MET is with respect to its PER, the more utilization is required of the processor. This means, as they get closer to being equal, the operator gets closer to needing a processor all to itself.

Static scheduling requires having advance knowledge of the process behavior. This means all operators must be effectively periodic [Ref. 12], and further means sporadic

operators must be converted so that they have *equivalent periods*. The process of attaining the equivalent period is somewhat complex, and involves heuristics. The outcome is never greater than min[(MRT - MET), MCP], and so this can be used as an upper bound. Using this information and other theorems presented in reference [Ref. 11], the following constraint is required of all sporadic operators.

$$\mathbf{2 \times MET \leq MRT,\ MET \leq MCP}$$

Presented thus far are the constraints that are imposed on otherwise valid time-critical operators. There are other timing constraints that are somewhat more intuitive.

First, every time critical operator must have an MET so that the scheduler can know how much time must be allocated to the operator in the schedule. Therfore, if is an error for an operator to have a PER, FW, MCP, or MRT without a MET.

Second, a time-critical operator must be periodic exclusive-or sporadic. This means it must be one or the other, but not both. Because of this, mixing constraints from both and having both is definitely an error.

Table 5 below shows the semantic checking that can be accomplished by the SDE. Again, cases not shown in the table are either trivial, calling for no conditional checking, or require sporadic operators to be converted. As SDE does not currently do conversion of sporadic operators for semantic checking, these can not be checked. They can be found, however, in reference [Ref. 11].

Y = Yes, is specified					N = No, is not specified
MET	PERIODIC		SPORADIC		Remarks
	PER	FW	MRT	MCP	
N	N	N	N	N	OK
N	N	N	N	Y	Error: Time-Critical Operators must have a MET
N	N	N	Y	N	Error: Time-Critical Operators must have a MET
N	N	N	Y	Y	Error: Time-Critical Operators must have a MET
N	N	Y	N	N	Error: Time-Critical Operators must have a MET
N	N	Y	N	Y	Error: Time-Critical Operators must have a MET
N	N	Y	Y	N	Error: Time-Critical Operators must have a MET
N	N	Y	Y	Y	Error: Time-Critical Operators must have a MET
N	Y	N	N	N	Error: Time-Critical Operators must have a MET
N	Y	N	N	Y	Error: Time-Critical Operators must have a MET
N	Y	N	Y	N	Error: Time-Critical Operators must have a MET
N	Y	N	Y	Y	Error: Time-Critical Operators must have a MET
N	Y	Y	N	N	Error: Time-Critical Operators must have a MET
N	Y	Y	N	Y	Error: Time-Critical Operators must have a MET
N	Y	Y	Y	N	Error: Time-Critical Operators must have a MET
N	Y	Y	Y	Y	Error: Time-Critical Operators must have a MET
Y	N	N	N	N	Warning: MRT and MCP will take default values
Y	N	N	N	Y	Warning: MRT will default to MCP + MET
Y	N	N	Y	N	Warning: MCP will default to MRT - MET
Y	N	N	Y	Y	If NOT(2*MET≤MRT, MET≤MCP) Then Error: Prototype will not schedule
Y	N	Y	N	N	Warning: PER will default to FW
Y	N	Y	N	Y	Error: Operators can not be both Periodic and Sporadic
Y	N	Y	Y	N	Error: Operators can not be both Periodic and Sporadic
Y	N	Y	Y	Y	Error: Operators can not be both Periodic and Sporadic
Y	Y	N	N	N	Warning: FW will default to PER
Y	Y	N	N	Y	Error: Operators can not be both Periodic and Sporadic
Y	Y	N	Y	N	Error: Operators can not be both Periodic and Sporadic
Y	Y	N	Y	Y	Error: Operators can not be both Periodic and Sporadic
Y	Y	Y	N	N	If NOT(MET ≤ FW ≤ PER) Then Error: Prototype will not schedule
Y	Y	Y	N	Y	Error: Operators can not be both Periodic and Sporadic
Y	Y	Y	Y	N	Error: Operators can not be both Periodic and Sporadic
Y	Y	Y	Y	Y	Error: Operators can not be both Periodic and Sporadic

Table 5. Semantic Timing Constrainst to be done by SDE.

C. MODIFICATIONS

Modifications to the SDE are based on the additional functionality outlined in the last section. That functionality was summarized in Tables 4 and 5.

1. Data Triggering Constraints

When looking at the remarks of the Table 4, it is apparent that the data triggering constraints can be further simplified into five major points:

1) When the producer and consumer operator are both periodic and connected by a dataflow stream, the PER of the producer must greater than or equal to that of the consumer,
2) When the producer of a stream is time-critical and the consumer is not, if the consumer has a BY ALL trigger, overflow can result,
3) When the producer of a stream is time-critical and the consumer is not, if the consumer does not have a BY SOME trigger, data loss can result.

There is some commonality about these rules that will be exploited in order to minimize the task of modifying the SDE. First, each rule identifies a consumer or producer (or both) of a stream. Therefore, a structure will be needed to store whether an operator is a producer or a consumer. It will also need to keep track of whether the consumer operators have a trigger and what type it is. The rules also depend on the knowledge of whether or not the operator is time-critical. This means a set of time-critical operators must be created so they can be kept track of separately. Further, those time-critical operators must be broken down into a mutually exclusive set of periodic and sporadic operators. And one more piece of information about those operators that is needed is the value of its PER. The astute reader will note that sporadic operators do not have a PER. This means that the sporadic operator should be converted so that it will have an equivalent PER.

Timing Constraints

An inspection of the remarks of Table 5 will also bring to light some commonality that can be summarized. Those rules can be broken down into six major points:

1) A time-critical operator must have a MET,
2) A time-critical operator can not mix periodic and sporadic timing constraints,
3) A time-critical operator with only one of its two timing constraints (FW and PER for Periodic and MRT and MCP for Sporadic) will have the unspecified

constraint set to a trivial, default value,

4) An otherwise valid periodic operator must follow the specification of MET≤FW≤PER or the prototype can not be scheduled,
5) An otherwise valid sporadic operator must follow the specification of 2*MET≤MRT and MCP ≤ MCP or the prototype can not be scheduled.

Just as with the data triggering rules, these must be able to identify whether an operator is constrained. In addition, they must be able to determine the existence and value of any FW, PER, MRT, and MCP constraints. The next step is to define structures to hold the information deemed required above.

2. Abstract Syntax Rule

Summarizing the requirements of both the data triggering rules and the timing constraint rules three structures will be required. The name of each structure is in bold parenthesis. An edge set (**edge_set**) is needed to keep track of all edges. It will store the edge's name, producer operator, consumer operator, and latency value. The latency is not needed for these rules, but it makes sense to keep it, as it could be used in future enhancements to the SDE. The second structure needed to support these rules is a constrained operator set (**op_id_met_set**). It will store the name of all operators that have a MET and the value of the MET, thereby identifying constrained operators. Finally, a structure will be needed to store all of the constrained operators and their constraints in a set (**id_constraint_set**). It will store the name of all constrained operators, their time constraints (per and fw for periodic operators, and mcp and mrt for sporadic operators), and type of trigger.

The last two structures, which will be referred to as phyla from here on, look as though they should be combined. The first attempt at defining additional abstract syntax rules did in fact have these two together. The problem is that the MET exists in a separate section of the derivation tree than do the other timing constraints. Therefore, these two distinct phyla need to exist at the lower levels of the tree anyway. The three additional phyla required are displayed in Figure 44 below. The phyla not terminating here are further defined in the full abstract syntax rules of PSDL in Appendix B as a phylum or lexeme. REAL is a built in type.

```
list op_id_met_set;
op_id_met_set
    : exported OpIdMetSetNil()
    | exported OpIdMetPair(op_id_met op_id_met_set);
op_id_met
    : exported OpIdMetNull()
    | exported OpIdMet(operator_id met);
met
    : exported MetNull()
    | exported MET(REAL);
/*-------------------------------------------------*/
list id_constraint_set;
id_constraint_set
    : exported IdConstraintSetNil()
    | exported IdConstraintPair(id_constraint id_constraint_set);
id_constraint
    : exported IdConstraintNull()
    | exported IdConstraint(operator_id time_constraints trigger);
time_constraints
    : exported OpConstraintsNull()
    | exported Periodic(per fw)
    | exported Sporadic(mcp mrt);
trigger
    : exported TriggerByNull()
    | exported TriggerByAll()
    | exported TriggerBySome();
per
    : exported PerNull()
    | exported Per(REAL);
fw
    : exported FwNull()
    | exported FW(REAL);
mcp
    : exported McpNull()
    | exported MCP(REAL);
mrt
    : exported MrtNull()
    | exported MRT(REAL);
/*-------------------------------------------------*/
list edge_set;
edge_set
    : exported EdgeSetNil()
    | exported EdgePair(edge edge_set);
edge
    : exported EdgeNull()
    | exported Edge(id latency producer consumer);
latency
    : exported LatencyNull()
    | exported Latency(REAL);
producer
    : exported ProducerNull()
    | exported Producer(operator_id);
consumer
    : exported ConsumerNull()
    | exported Consumer(operator_id);
```

Figure 44. Additional Abstract Syntax Rules for SDE.

As tree representations are sometimes more familiar and easier to understand, these rules have been re-displayed in the trees of Figure 45 below. While it doesn't contain as much information as the actual specifications, it may make quickly referencing the rules easier.

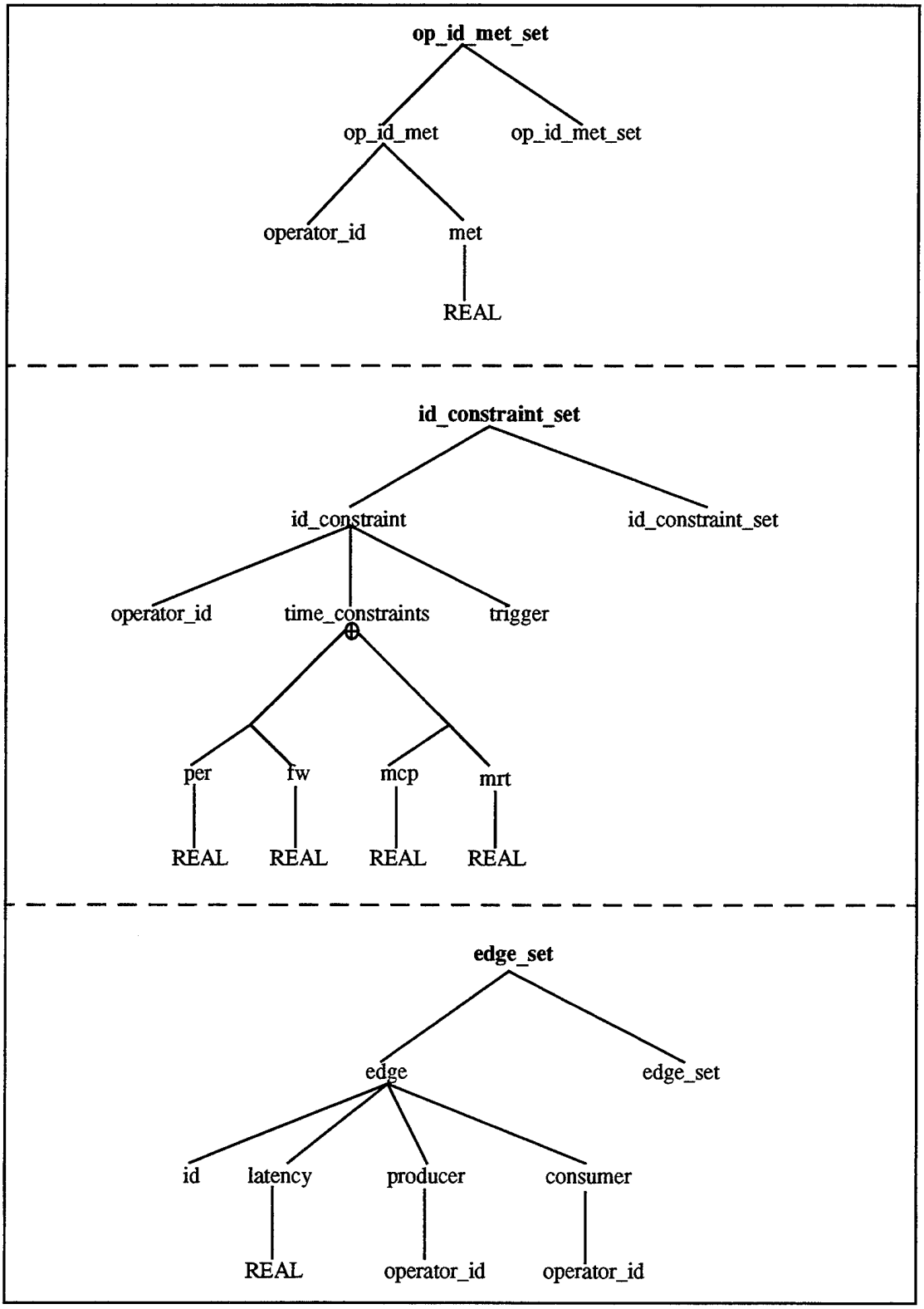


Figure 45. Addition Abstract Syntax Rules for SDE (2).

3. Attribute Rules

All of the timing constraints ended up being represented as real numbers as opposed to the normal representation of integers with an unit (i.e. 90 milliseconds). These were represented as real so that no matter what number is given in the design, it could be converted to a default internal representation for comparison purposes. This means that when the auxiliary functions were defined (Appendix E), there had to be one that converts the timing constraint representation within the old abstract syntax to real for compliance with this abstract syntax.

With one exception, the attribute names are simply the same as the newly created phylum with the prefix syn or inh, depending on whether it is synthesized or inherited. The exception is the syn_vertex_id_met_set of type op_id_met_set. The word "vertex" was substituted in for "op" because, while they are the same thing, the previously defined abstract grammar used the term vertex. Therefore, the declared attribute name was changed.

Earlier, it was stated that the MET and other timing constraints were established in different parts of the derivation tree. Figure 46 is a partial look at the abstract syntax rules for the PSDL SDE in tree form. On the tree the attributes equations required for the semantic checking outlined above are shown in bold type. Note where the MET information is located as opposed to where the timing constraint information is (typed within parenthesis at the bottom of the figure). The MET is defined under a_vertex, the edge constraints under an_edge, and the timing constraints under a_constraint. Therefore, that is where the synthesized attributes had to start. While the synthesized attributes bubble up the tree, the inherited attributes sink down the tree. This is evident when looking at the Figure 46. The vertex_id_met_set bubbled up starting at the vert_list phylum, continuing on to the graph phylum and then on to the operator_impl phylum where it is copied to the inh_vertex_id_met_set attribute at the operator_impl phylum. From there, it begins sinking on the right side of the tree; first to the cc phylum and then to the constraints phylum where it can be used for semantic checking.

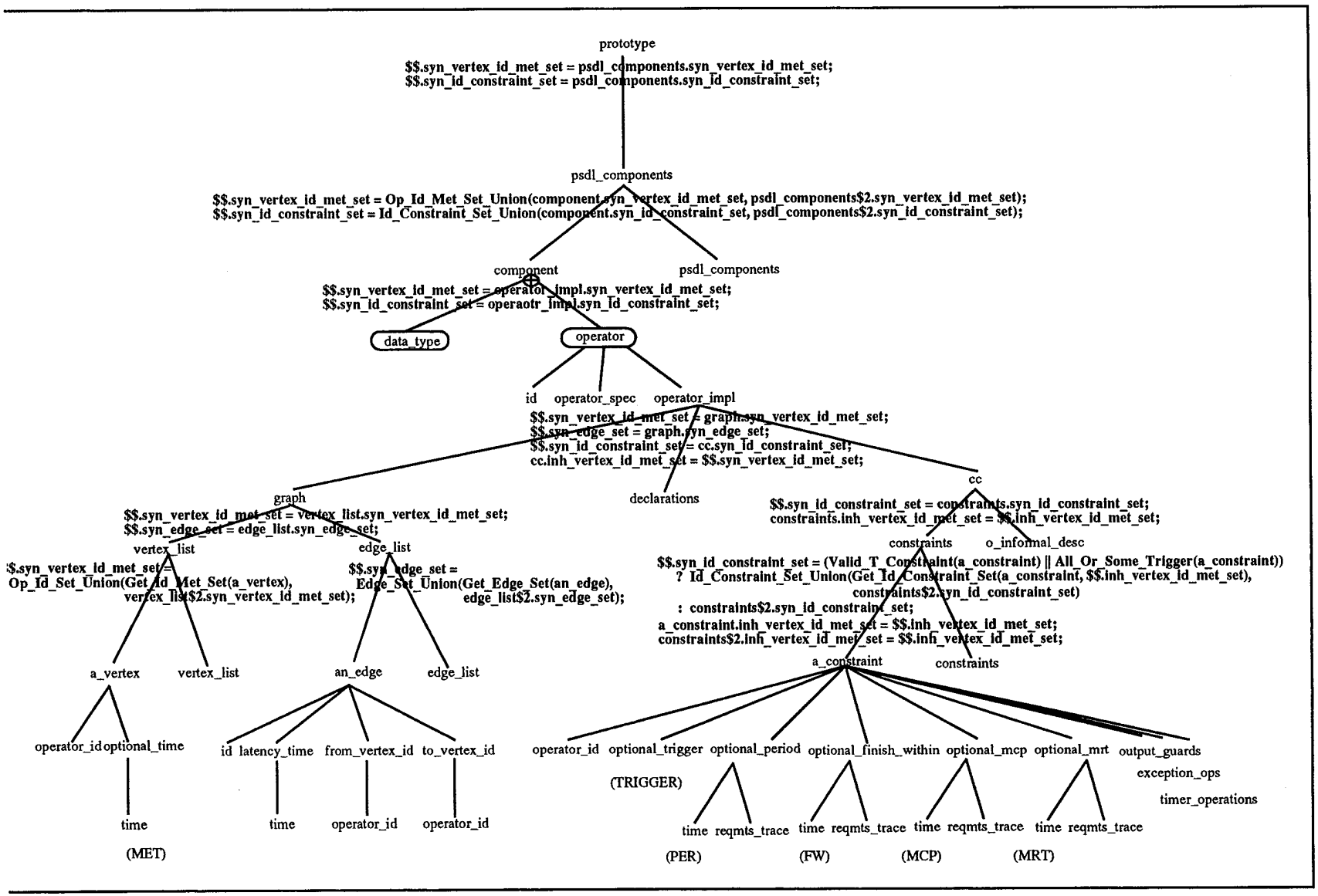


Figure 46. Abstract Syntax with Attributes Equations.

While it seems a little backwards to show the attribute equations before the attribute declarations, it was done to not only demonstrate what the attribute equations are but more importantly show graphically where they must exist.

Recall that all synthesized attributes are built at the lower levels and then passed up the tree. Also, realize that in lists, every set of attributes, which are defined by the equations in the figure, will exist once for each occurrence in that list. Psdl_components, for example, is composed of a component and another psdl_components (psdl_components$2). The equations under the phylum component (shown in the figure) will synthesize all of its attributes for the mentioned component (shown in the figure) and again for every other component in psdl_components$2, recursively. Therefore, at the component phylum, the set of id_constraints, for example, will be only the ones within that component (its children). At the prototype phylum, however, there will be one set and it will contain all of the id_constraints within the prototype.

The edge_set didn't need to be collected into one single set for all of them, so it was not synthesized up to the prototype phylum. It stopped at the operator_impl phylum, which then had all of the edges for its child operators. In fact, the semantic checking requires all of the attributes to be collected at the operator_impl phylum. After all, the best place to check and subsequently notify the designer of a problem is at the component where the problem exists. The reason that the other two attributes (id_met_set and id_constraint_set) are synthesized all the way up to the prototype phylum is so that the number of processors required could be determined. It was decided that the Load Factor equation would be used to determine how many processors are required by the design. This, of course, necessitates having all of the MET's and PER's in one place. What this also calls for, is being able to convert the sporadic operators. Because there was not enough time to do that function, the number of processors required is only based on periodic operators. The display on the editor must state this disclaimer.

Figure 47 below shows both the attribute declarations and equations for the synthesized and inherited attributes. The local attributes and equations for semantic checking will be discussed next along with the required additions to the attribute rules.

```
prototype, psdl_components, component {syn op_id_met_set     syn_vertex_id_met_set;};

prototype, psdl_components, component {syn id_constraint_set syn_id_constraint_set;};
operator_impl, graph, vertex_list     {syn op_id_met_set     syn_vertex_id_met_set;};
operator_impl, graph, edge_list       {syn edge_set          syn_edge_set;};
operator_impl, cc, constraints        {syn id_constraint_set syn_id_constraint_set;};
cc, constraints, a_constraint         {inh op_id_met_set     inh_vertex_id_met_set;};

prototype
  : Prot
    {$$.syn_vertex_id_met_set = psdl_components.syn_vertex_id_met_set;
     $$.syn_id_constraint_set = psdl_components.syn_id_constraint_set;};
psdl_components
  : PsdlNil
    {$$.syn_vertex_id_met_set = OpIdMetSetNil;
     $$.syn_id_constraint_set = IdConstraintSetNil;}
  | PsdlPair
    {$$.syn_vertex_id_met_set = Op_Id_Met_Set_Union(component.syn_vertex_id_met_set,
                                                    psdl_components$2.syn_vertex_id_met_set);
     $$.syn_id_constraint_set = Id_Constraint_Set_Union(component.syn_id_constraint_set,
                                                        psdl_components$2.syn_id_constraint_set);};
component
  : NoComponent
    {$$.syn_vertex_id_met_set = OpIdMetSetNil;
     $$.syn_id_constraint_set = IdConstraintSetNil;}
  | Op
    {$$.syn_vertex_id_met_set = operator_impl.syn_vertex_id_met_set;
     $$.syn_id_constraint_set = operator_impl.syn_id_constraint_set;}
  | Data
    {$$.syn_vertex_id_met_set = OpIdMetSetNil;
     $$.syn_id_constraint_set = IdConstraintSetNil;};
operator_impl
  : OpImplNull, AdaOpImpl
    {$$.syn_vertex_id_met_set = OpIdMetSetNil;
     $$.syn_edge_set          = EdgeSetNil;
     $$.syn_id_constraint_set = IdConstraintSetNil;}
  | OperatorImpl
    {$$.syn_vertex_id_met_set = graph.syn_vertex_id_met_set;
     $$.syn_edge_set          = graph.syn_edge_set;
     $$.syn_id_constraint_set = cc.syn_id_constraint_set;
     cc.inh_vertex_id_met_set = $$.syn_vertex_id_met_set;};
graph
  : GraphNull
    {$$.syn_vertex_id_met_set = OpIdMetSetNil;
     $$.syn_edge_set          = EdgeSetNil;}
  | Graph
    {$$.syn_vertex_id_met_set = vertex_list.syn_vertex_id_met_set;
     $$.syn_edge_set          = edge_list.syn_edge_set;};
vertex_list
  : VertexListNull
    {$$.syn_vertex_id_met_set = OpIdMetSetNil;}
  | VertexListPair
    {$$.syn_vertex_id_met_set = Op_Id_Met_Set_Union(Get_Id_Met_Set(a_vertex),
                                                    vertex_list$2.syn_vertex_id_met_set);};
edge_list
  : EdgeListNil
    {$$.syn_edge_set = EdgeSetNil;}
  | EdgeListPair
    {$$.syn_edge_set = Edge_Set_Union(Get_Edge_Set(an_edge),
                                      edge_list$2.syn_edge_set);};
cc
  : CcNull
    {$$.syn_id_constraint_set = IdConstraintSetNil;}
  | Cc
    {$$.syn_id_constraint_set = constraints.syn_id_constraint_set;
     constraints.inh_vertex_id_met_set = $$.inh_vertex_id_met_set;};
constraints
  : ConstraintsNull
    {$$.syn_id_constraint_set = IdConstraintSetNil;}
  | ConstraintsPair
    {$$.syn_id_constraint_set = (Valid_T_Constraint(a_constraint) || All_OR_Some_Trigger(a_constraint))
          ? Id_Constraint_Set_Union(Get_Id_Constraint_Set(a_constraint, $$.inh_vertex_id_met_set),
                                    constraints$2.syn_id_constraint_set)
          : constraints$2.syn_id_constraint_set;
     a_constraint.inh_vertex_id_met_set  = $$.inh_vertex_id_met_set;
     constraints$2.inh_vertex_id_met_set = $$.inh_vertex_id_met_set;};
```

Figure 47. Synthesized and Inherited Attribute Equations of Attribute Rules.

Because the amount of code for implementing the local attributes, equations and auxiliary functions for the semantic checking required is large, only part of the changes to the

SDE will be shown here. Appendix C contains a full listing of the PSDL SDE Attribute Rules, including those added by this thesis. The additions to the operator_impl phylum are significant enough to show some of the work done. It is represented in Figure 48.

```
operator_impl
  : OpImplNull, AdaOpImpl{}
  | OperatorImpl
    {local BOOL producerop_period_le_consumerop_error;
     producerop_period_le_consumerop_error =
       (Is_ProducerOp_Period_LE_ConsumerOp($$.syn_edge_set,
                                           $$.syn_id_constraint_set));
     local STR producerop_period_le_consumerop_msg;
     producerop_period_le_consumerop_msg = (producerop_period_le_consumerop_error)
          ?  "\n"
           # "-- !For any edge, the Producer's Period\n"
           # "--  must exceed that of the Consumer."
          :  "";
/*-------------------*/

       local BOOL sporadic_consumerop_wo_trigger_error;
       sporadic_consumerop_wo_trigger_error =
         (Is_Sporadic_ConsumerOp_WO_Trigger($$.syn_edge_set,
                                            $$.syn_id_constraint_set));

       local STR sporadic_consumerop_wo_trigger_msg;
       sporadic_consumerop_wo_trigger_msg =
         (sporadic_consumerop_wo_trigger_error)
         ?  "\n"
          # "-- !For any edge, if the Consumer is\n"
          # "--  Sporadic, it must have a Trigger."
         :  "";

/*-------------------*/
       local BOOL constr_producerop_and_unconstr_consumerop_w_trigger_error;
       constr_producerop_and_unconstr_consumerop_w_trigger_error =
         (Is_Constr_ProducerOp_And_Unconstr_ConsumerOp_W_Trigger($$.syn_edge_set,
                                                                 $$.syn_vertex_id_met_set,
                                                                 $$.syn_id_constraint_set));
       local STR constr_producerop_and_unconstr_consumerop_w_trigger_msg;
       constr_producerop_and_unconstr_consumerop_w_trigger_msg =
         (constr_producerop_and_unconstr_consumerop_w_trigger_error)
         ?  "\n"
          # "-- !For any edge, if the Producer is constrained,\n"
          # "--  an unconstrained Consumer can not have a Trigger."
         :  "";
/*-------------------*/
       local BOOL unconstr_producerop_and_constr_consumerop_w_byall_error;
       unconstr_producerop_and_constr_consumerop_w_byall_error =
         (Is_Unconstr_ProducerOp_And_Constr_ConsumerOp_W_ByAll($$.syn_edge_set,
                                                               $$.syn_vertex_id_met_set,
                                                               $$.syn_id_constraint_set));
       local STR unconstr_producerop_and_constr_consumerop_w_byall_msg;
       unconstr_producerop_and_constr_consumerop_w_byall_msg =
         (unconstr_producerop_and_constr_consumerop_w_byall_error)
         ?  "\n"
          # "-- !For any edge, if the Producer is unconstrained,\n"
          # "--  a constrained consumer triggered By_All\n"
          # "--  can result in Overflow."
         :  "";
/*-------------------*/
       local BOOL unconstr_producerop_and_constr_consumerop_w_bysome_error;
       unconstr_producerop_and_constr_consumerop_w_bysome_error =
         (Is_Unconstr_ProducerOp_And_Constr_ConsumerOp_W_BySome($$.syn_edge_set,
                                                                $$.syn_vertex_id_met_set,
                                                                $$.syn_id_constraint_set));
       local STR unconstr_producerop_and_constr_consumerop_w_bysome_msg;
       unconstr_producerop_and_constr_consumerop_w_bysome_msg =
         (unconstr_producerop_and_constr_consumerop_w_bysome_error)
         ?  "\n"
          # "-- !For any edge, if the Producer is unconstrained,\n"
          # "--  a constrained consumer triggered By_Some\n"
          # "--  can result in Data loss."
         :  "";
/*-------------------*/
       local BOOL has_error;
       has_error =
         (producerop_period_le_consumerop_error                    ||
         sporadic_consumerop_wo_trigger_error                      ||
         constr_producerop_and_unconstr_consumerop_w_trigger_error ||
         unconstr_producerop_and_constr_consumerop_w_byall_error   ||
         unconstr_producerop_and_constr_consumerop_w_bysome_error);
       local STR error_header;
       error_header = (has_error)
                      ? "\n"
                       #"-----------------------------------------\n"
                       #"-- SCHEDULING NOTICE:"
                      : "";
       local STR error_trailer;
       error_trailer = (has_error)
                       ? "\n"
                        #"-----------------------------------------"
                       : "";
    };
```

Figure 48. Local Attributes and Equations for operator_impl Phylum.

The code is fairly straight forward and separated into sections divided by dashed lines. Which section implements which semantic check can be quickly determined by looking at one of the two possible string values that are assigned. Each correspond with the remarks of Table's 4 & 5, and are mentioned again at the beginning of this section. The last section in Figure 48 merely sets up the header and trailer if any of the previous constraint checking variables are true (indicating violation).

Many functions are referenced in the last two figures. The previously existing auxiliary functions are listed in Appendix D, while the auxiliary functions written in support of this thesis are in Appendix E. Recall that function names, for the conventions here, are mixed upper and lower case with underscoring; i.e. This_Is_A_Function_Name.

To ensure that the functions are understood, one will be covered here. The first section of code in the last figure checks to ensure that for any edge, if the consumer operator's period is greater than or equal to the producer's, an appropriate error message is displayed. The boolean variable is set to true, setting the displayed string to the error message vice null, if the function Is_ProducerOP_Period_LE_ConsumerOP returns true. This is a boolean function that takes two arguments (operator_impl.syn_edge_set, operator_impl.syn_id_constraint_set), and is displayed in Figure 49.

```
BOOL exported
Is_ProducerOp_Period_LE_ConsumerOp(edge_set          es,
                                   id_constraint_set cs)  (
   with(es)  (
      EdgeSetNil:           false,
      EdgePair(hd, tl):
      with(hd)  (
         EdgeNull:         Is_ProducerOp_Period_LE_ConsumerOp(tl, cs),
         Edge(*, *, p, c):
         with(p)  (
            ProducerNull:     Is_ProducerOp_Period_LE_ConsumerOp(tl, cs),
            Producer(p_opid):
            with(c)  (
               ConsumerNull:     Is_ProducerOp_Period_LE_ConsumerOp(tl, cs),
               Consumer(c_opid):
               (Get_Period(p_opid, cs) == PerNull ||
                Get_Period(c_opid, cs) == PerNull    )
               ? Is_ProducerOp_Period_LE_ConsumerOp(tl, cs)
               : (Get_Period(p_opid, cs) <= Get_Period(c_opid, cs))
                 ? true
                 : Is_ProducerOp_Period_LE_ConsumerOp(tl, cs)
            )
         )
      )
   )
);
```

Figure 49. Auxiliary Function.

As mentioned in chapter three, if not familiar with recursion, this would be a good time to review. Most of the functions take on this same format. The keyword "with" is used like a case statement that covers all of the operators of the with'd phylum, and most of them

recursively search sets such as the edge_set or id_constraint_set. When looking at a function such as this, ensure that the phylum description (abstract syntax) of the attributes passed in are close by. This makes reading the function much easier.

The displayed function first looks at the edge_set. If it is nil (empty), the answer is false (there is no edge in error). If the set is not empty, the first edge or the head (an_edge) is pulled off and its producer and consumer operator_id's are determined. Then, the function Get_Period is used with those operator_id's to return the period's of each and are compared to see if either is null. If that is the case, or if a nullary value is hit prior to this point, the function is simply called again with the same id_constraint_set and the rest of, or tail of the edge_set. If there are two valid periods assigned, they are checked to ensure that the producer's exceeds the consumers. If not, true is returned (thus an error). If so, once again, the function is called with the tail of the edge_list. This will continue until an error is reached or until the end of the list is reached when EdgeSetNil is found and false is returned.

4. Unparsing Rules

The unparsing rules only called for a few additions. Recall, theses are the specifications that control what is displayed by the editor. The only items that must be displayed and weren't already are the additional local string attributes that were defined in the attribute rules to either display the error that exists or a null value when no error exists.

Figure 50 shows the unparsing for the operator_impl phylum because that is the one displayed with its attributes in the last figure. Notice that all of the local string attributes declared and built in the attribute rules are displayed under the operator named OperatorImpl. The various views represent different unparsing schemes so that what is seen by the user depends on which view is being displayed and multiple views can be displayed at once if desired. Also notice that comments are placed after each placeholder to help identify which phylum is represented by that placeholder.

```
operator_impl
  : OpImplNull
    [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @:::="%n<operator implementation>"]
  | AdaOpImpl
    [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,IMPL_ONLY_VIEW
          @:::="%nIMPLEMENTATION ADA " @ "%n%t" "%b%nEND"]
  | OperatorImpl
    [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @:::="%nIMPLEMENTATION"
          @     /*graph*/
       error_header
       producerop_period_le_consumerop_msg
       sporadic_consumerop_wo_trigger_msg
       constr_producerop_and_unconstr_consumerop_w_trigger_msg
       unconstr_producerop_and_constr_consumerop_w_byall_msg
       unconstr_producerop_and_constr_consumerop_w_bysome_msg
       error_trailer
            ^     /*declarations*/
            @     /*cc*/
            "%nEND"]
     [BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW @::= "%nIMPLEMENTATION"
            @     /*graph*/
            ^     /*declarations*/
            @     /*cc*
            "%nEND"]
  ;
```

Figure 50. Unparsing Rules for operator_impl Phylum.

5. Transformation Rules

The transformation rules of the SDE were not modified by this thesis and therefore are not discussed here. Appendix F has the transformation rules for the PSDL SDE.

6. Concrete Rules

The concreted rules were not modified either and therefore are not discussed but are fully listed in Appendix G.

D. TESTING

In the beginning of this chapter, a prototype was developed to give an example on how to use the SDE. Lets see how the new editor reacts with the same prototype. Particularly of interest is how the editor reacts when entering the timing constraints. For that reason, Figure 36 is repeated below.

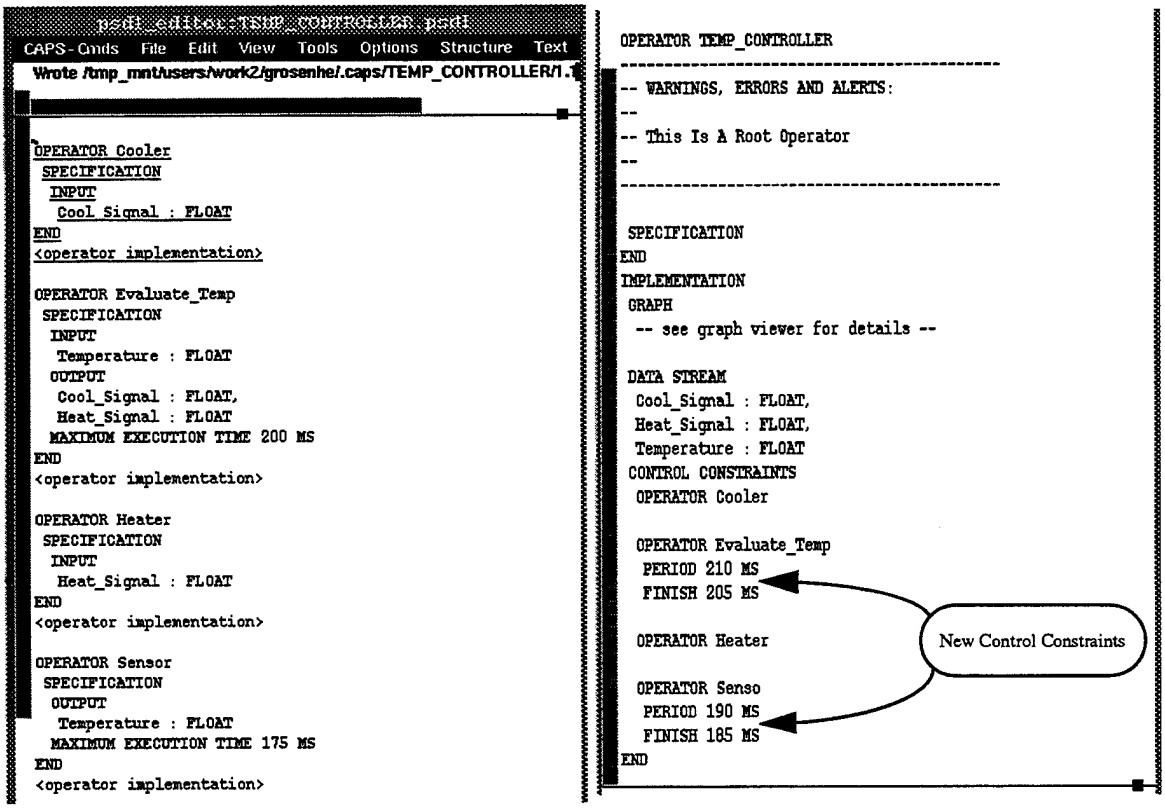


Figure 36. Complete PSDL Program with Control Constraints (Repeated).

The same constraints will be inserted into the prototype just created. Notice in Figure 51 that when inputting the PER for operator Evaluate_Temp, a message comes up and states that if left unspecified, FW will default to PER. It will do this for PER, FW, MRT, and MCP.

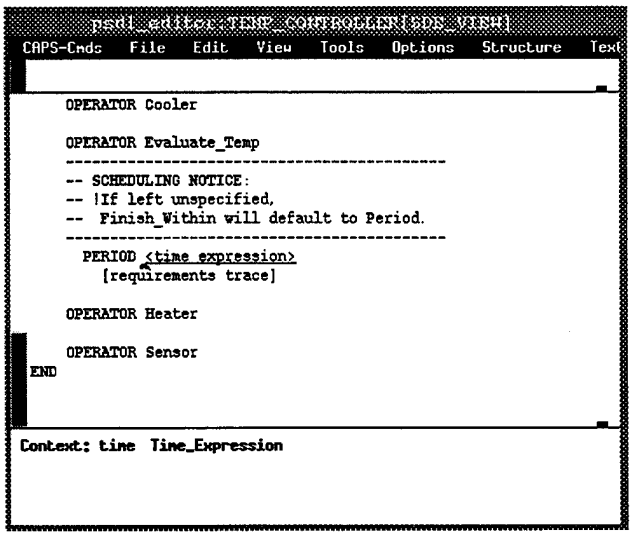


Figure 51. SDE with Default Message.

Going further, the PER and FW is input for the Evaluate_Temp operator. So far, so good. The editor does not complain, so the Sensor operator is given the PER constraint defined earlier. As soon as the return key is hit, as Figure 52 indicates, the editor gives the familiar default message, but there is another message. The new scheduling notice states that the PER of the producer of a stream must exceed that of the consumer. This constraint, under the old editor, would have gone un-noticed until translated and scheduled, thereby eating up a great deal of time. Not to mention the fact that the designer may no longer have this information fresh in his or her memory.

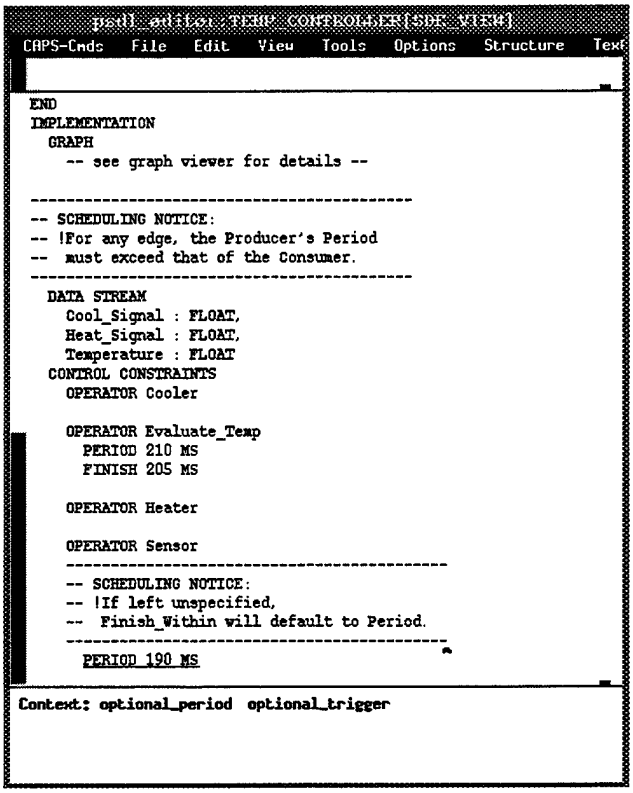


Figure 52. SDE with PER Constraint Error.

But wait, there is another message. This prototype, because of the way the constraints were assigned will not run on a uniprocessor. The message at the top of the editor, based on the Load Factor discussed earlier, states that the prototype will require at least 2 processors to schedule. Apparently each operator was constrained in such a way that they both will require a processor of their own. Figure 53 shows the message at the top of the editor. This illustrates a very good point made earlier. As discussed in the beginning of this thesis, scheduling hard real-time systems is very difficult. If this system were real and had to run on a single CPU, we

would have already blown it. The designer needs quick feedback as to whether the design will work or not. If this isn't provided, a great deal of time could be spent on a design that is impossible to implement, before the designer realizes it can't work. The other semantic checks display error messages such as the ones shown thus far. All of the checks outlined in Table 4 & 5 earlier have been implemented and their message reads very much like the remarks of the tables. It is up to the reader to experiment with the SDE to see all of them for one self.

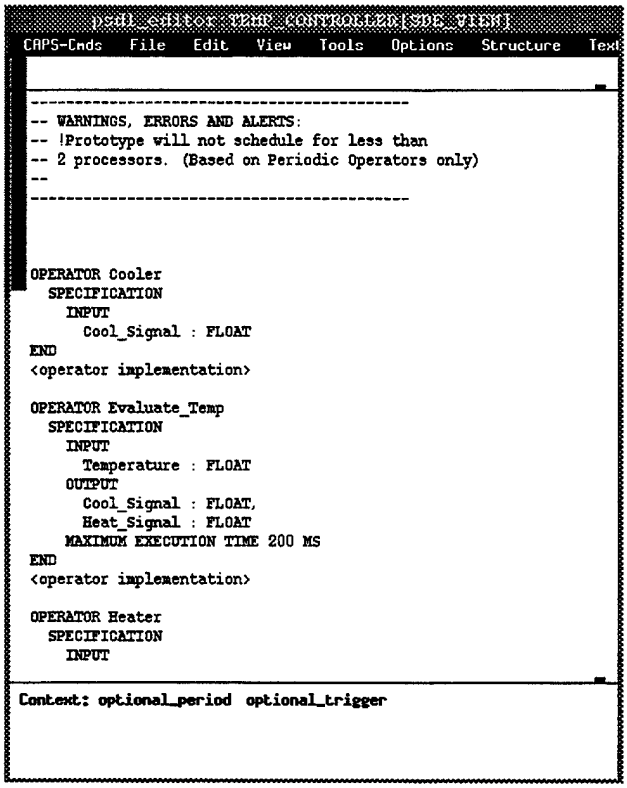


Figure 53. SDE with number processors required message.

V. FUTURE RESEARCH AND DEVELOPMENT

One of the biggest obstacles to new tools is acceptance by new, potential users. The learning curve plays a major part of this. If the tool is too difficult to use, no matter how good it is, most will reject it. For this reason, more and more time is being spent on developing good user interfaces as well as developing good tools.

Learning how to use the SDE not a trivial task and many are intimidated by it at first. The graphical editor, however, comes somewhat natural to most users. The CAPS tool-set would be embraced more strongly if the input functionality was moved to the graphical editor. Users would not have to learn to use a new editor nor would they have to learn a new language (PSDL). Information could be taken from and presented to the designer in a more intuitive fashion.

The information that is currently input via the graphical editor is propagated to the SDE. Research should to be done to determine how much more of this propagation can be achieved going in both directions between the derivation tree of the SDE and the graphical editor. The goal should be such that not only will the Graphical Editor account for all (or as much as possible) of the input, but well formed messages should be displayed back to the designer while still in the Graphical Editor. A separate window, perhaps utilizing TAE+, could be designed for displaying both the feedback messages that currently exist within the SDE and those of future implementations.

The display of how many processors are required to schedule the prototype can be particularly useful to the designer. Because the SDE can not convert the sporadic operators to have an equivalent period, it is not as accurate as could be. Some research should be devoted to whether or not this can be effectively done within the editor.

Additional checking that could be accomplished with the attributes already gathered include testing for constrained operators that have unconstrained children. Also, using the edges and operator MET's, each thread under a parent operator could be checked to ensure its total required execution time does not exceed that of the parent.

Other checking that can be accomplished is the verification of streams that are built during decomposition. For example if a parent operator has an external stream coming in and two regular streams coming out, when it is decomposed, the graph editor shows those

connections at the bottom as a reminder. If the designer doesn't include them or puts additional incoming or outgoing streams in, there are no warnings upon returning to the SDE. This inconsistency should not be allowed and can be tracked with the current edge attributes captured.

Many of the templates displayed in the help pane (menu window) should not be there. For example, the designer could pick psdl_implementation out of the help pane to replace the placeholder <operator implementation>. This does not work well and really shouldn't be an option to the designer. If the operator is to be implemented with PSDL, that should be done within the graphical editor. The same holds for input and output stream. It is more confusing to the new users of the SDE to have those displayed. Research should be done on what parts of the display can be removed for a more clear implementation of the editor.

LIST OF REFERENCES

1. Pressman, R., *SOFTWARE ENGINEERING A PRACTITIONER'S APPROACH*, McGraw-Hill, Inc., New York, St. Louis, San Fransisco,..., 1992.

2. Goldsack, S., Finkelstein, A., *Requirements engineering for real-time systems*, Software Engineering Journal, May, 1991.

3. Berzins, V., Luqi, Shing, M., Chmura, L., *Managing real-time Software Projects: Problems and Issues*, Technical Report NPSCS-92-016, Monterey Ca., December 1992.

4. Yourdon, E., *Modern Structured Analysis, Yourdon Press*, Englewood Cliffs, NJ, 1989.

5. Luqi, *Computer-aided software protoyping*, IEEE Computer, pp111-112, September, 1991.

6. Brockett, A CAPS Tutorial, Naval Postgraduate School, Monterey, Ca., 1995.

7. Bieman, J., Gordon, V., *Rapid Rototyping: Lessons Learned*, IEEE Software, pp. 85-95, January, 1995.

8. Luqi, *Software Evolution Via Prototyping*, Technical Report NPS52-88-039, Naval Postgraduate School, Monterey, Ca., September 1988

9. Luqi, *Software Evolution Through Rapid Prototyping*, IEEE Computer, pp. 13-25, May, 1989.

10. Berzins, V., Luqi, Yeh, R., *A Prototyping Language for Real-Time Software*, IEEE Transactions on Software Engineering, Vol 14, No 10, October 1988.

11. Cordeiro, M., *Disributed Hard Real-Time Scheduling for a Software Prototyping Environment*, Ph.D. Dissertation, Naval Postgraduate School, Monterey, Ca., March 1995.

12. Burns, A., *Scheduling hard real-time systems: a review*, Software Engineering Journal, pp. 116-127, May 1991.

13. Reps, T., and Teitelbaum, T., *The Synthesizer Generator, A system For Constructing Language-Based Editors*, Springer-Verlag, New York Inc., 1989.

14. White, L., *The Development Of A Rapid Prototyping Environment*, Master's Thesis, Naval Postgraduate School, Monterey, Ca., December, 1989.

15. *The Synthesizer Generator Reference Manual*, Release 4.1, Gramma Tech, Inc., Ithaca, NewYork.

APPENDIX A - PSDL Grammar

```
psdl
  = (component)

component
  = data_type
  | operator

data_type
  = "type" id type_spec type_impl

type_spec
  = "specification" ["generic" type_decl] [type_decl]
    ("operator" op_name operator_spec)
    [functionality] "end"

operator
  = "operator" op_name operator_spec operator_impl

operator_spec
  = "specification" (interface) [functionality] "end"

interface
  = attribute [reqmts_trace]

attribute
  = "generic" type_decl
  | "input" type_decl
  | "output" type_decl
  | "states" type_decl "initially" initial_expression_list
  | "exceptions" id_list
  | "maximum execution time" time

type_decl
  = id_list ":" type_name ("," id_list ":" type_name)

type_name
  = id
  | id "[" type_decl "]"

id_list
  = id ("," id)

reqmts_trace
  = "required by" id_list

functionality
  = [keywords] [informal_desc] [formal_desc]

keywords
  = "keywords" id_list

informal_desc
  = "description" "(" text ")"

formal_desc
  = "axioms" "(" text ")"

type_impl
  = "implementation ada" id "end"
  | "implementation" type_name ("operator" op_name operator_impl) "end"

operator_impl
  = "implementation ada" ada_op_name "end"
  | "implementation" psdl_impl "end"

psdl_impl
  = data_flow_diagram [streams] [timers] [control_constraints]
    [informal_desc]

data_flow_diagram
  = "graph" (vertex) (edge)

vertex
  = "vertex" op_id [":" time]
    -- time is the maximum execution time

edge
  = "edge" id [":" time] op_id "->" op_id
    -- time is the latency

op_id
  = [id "."] op_name ["(" [id_list] "|" [id_list] ")"]

streams
  = "data stream" type_decl

timers
  = "timer" id_list

control_constraints
  = "control constraints" constraint (constraint)

constraint
  = "operator" op_id
    ["triggered" [trigger] ["if" expression] [reqmts_trace]]
    ["period" time [reqmts_trace]]
    ["finish within" time [reqmts_trace]]
    ["minimum calling period" time [reqmts_trace]]
    ["maximum response time" time [reqmts_trace]]
    (constraint_options)

constraint_options
  = "output" id_list "if" expression [reqmts_trace]
  | "exception" id ["if" expression] [reqmts_trace]
  | timer_op id ["if" expression] [reqmts_trace]

trigger
  = "by all" id_list
  | "by some" id_list

timer_op
  = "reset timer"
  | "start timer"
  | "stop timer"

initial_expression_list
  = initial_expression ("," initial_expression)

initial_expression
  = "true"
  | "false"
  | integer_literal
  | real_literal
  | string_literal
  | id
  | type_name "." op_name ["(" initial_expression_list ")"]
```

```
  | "(" initial_expression ")"
  | initial_expression binary_op initial_expression
  | unary_op initial_expression

binary_op
  = "and" | "or" | "xor"
  | "<" | ">" | "=" | ">=" | "<=" | "/="
  | "+" | "-" | "&" | "*" | "/" | "mod" | "rem" | "**"

unary_op
  = "not" | "abs" | "-" | "+"

time
  = integer_literal unit

unit
  = "microsec"
  | "ms"
  | "sec"
  | "min"
  | "hours"

expression_list
  = expression ("," expression)

expression
  = "true"
  | "false"
  | integer_literal
  | time
  | real_literal
  | string_literal
  | id
  | type_name "." op_name ["(" expression_list ")"]
  | "(" expression ")"
  | initial_expression binary_op initial_expression
  | unary_op initial_expression

op_name
  = ada_op_name "_" integer_literal

ada_op_name
  = id "_" integer_literal

id
  = letter {alpha_numeric}

real_literal
  = integer_literal "." integer_literal

integer_literal
  = digit {digit}

string_literal
  = """ {char} """

char
  = any printable character except ")"

digit
  = "0 .. 9"

letter
  = "a .. z"
  | "A .. Z"
  | "_"

alpha_numberic
  = letter
  | digit

text
  = {char}
```

APPENDIX B - Abstract Rules

```
root prototype;

/* TOKEN DEFINITIONS */

/* definitions for unformatted text */
CLINEBREAK:     CLinebreakLex<<NO_WHITESPACE>[\n] >;
CLINE:          CLineLex< <NO_WHITESPACE> [^\n\)]+ >;
LCURLY:         LCurly<[\(] <NO_WHITESPACE> >;
RCURLY:         RCurly< [\)] <INITIAL> >;
WHITESPACE:     WhiteSpaceLex<[\ \t\n] >;
/* ------------------------------------------------------------*/

DESCRIPKW:      DescripKwLex<"DESCRIPTION" >
                |<"description" >;
INPUTKW:          <"INPUT" >
                | <"input" >;
OUTPUTKW:         <"OUTPUT" >
                | <"output" >;
EXCEPTIONSKW:   <"EXCEPTIONS" >
                | <"exceptions" > ;
EXCEPTIONKW:    <"EXCEPTION" >
                | <"exception" >;
AXIOMKW:        <"AXIOMS" >
                | <"axioms" >;
ADAKW:          <"ADA" >
                |<"ada" >;
INTKW:          <"INTEGER" >
                |<"integer" >;
REALKW:         <"REAL" >
                |<"real" >;
BOOLKW:         <"BOOLEAN" >
                |<"boolean" >;
TRUEKW:         <"TRUE" >
                |<"true" >;
FALSEKW:        <"FALSE" >
                |<"false" >;
MICRO:          <"MICROSEC" >
                |<"microsec" >;
MS:             MSLex<"MS" >
                | <"ms" >;
SEC:            SECLex<"SEC" >
                |<"sec" >;
MIN:            MINLex<"MIN" >
                |<"min" >;
HOURS:          HOURSLex<"HOURS" >
                |<"hours" >;
NOTKW:          NOTLex<"NOT" >
                |<"not" >;
ANDKW:          AndLex<"AND" >
                |<"and" >;
ABSKW:          ABSLex<"ABS" >
                |<"abs" >;
GTEKW:          GTELex<">=" >;

LTEKW:          LTELex<"<=" >;

NEQVKW:         NEQVlex<"/=" >;

REMKW:          REMLex<"REM" >
                |<"rem" >;
MODKW:    MODLex<"MOD" >
      |<"mod" >;
EXPKW:          EXPLex<"EXP" >
                |<"exp" >
      |<"**" >;
ENDKW:          ENDLex<"END" >
                |<"end" >;
TYPEKW:         TypeLex<"TYPE" >
                |<"type" >;
OPERATORKW:     OperatorLex<"OPERATOR" >
                |<"operator" >;
SPECKW:         SpecificationLex<"SPECIFICATION" >
                |<"specification" >;
GENERICKW:      GenericLex<"GENERIC" >
                |<"generic" >;
REQBYKW:        ReqByLex<"REQUIRED\ BY" >
                |<"required\ by" >;
STATESKW:       StatesLex<"STATES" >
                |<"states" >;
INITIALKW:      InitiallyLex<"INITIALLY" >
                |<"initially" >;
ORKW:           OrLex<"OR" >
                |<"or" >;
XORKW:          XorLex<"XOR" >
                |<"xor" >;
KEYWKW:         KeyWordsLex<"KEYWORDS" >
                |<"keywords" >;
IMPLKW:         ImplementationLex<"IMPLEMENTATION" >
                |<"implementation" >;
GRAPHKW:        GraphLex<"GRAPH" >
                |<"graph" >;
VERTEXKW:       VertexLex<"VERTEX" >
                |<"vertex" >;
ARROWKW:        ArrowLex<"->" >;
EDGEKW:         EdgeLex<"EDGE" >
                | <"edge" >;
DATSTRKW:       DataStreamsLex<"DATA\ STREAM" >
                |<"data\ stream" >;
TIMERKW:        TimerLex<"TIMER" >;

CONTROLKW:      ControlLex<"CONTROL\ CONSTRAINTS" >
                |<"control\ constraints" >;
TRIGGERBYKW:    TriggerByLex<"TRIGGERED\ BY" >
                |<"triggered\ by" >;
TRIGGERKW:      TriggerLex<"TRIGGERED" >
                |<"triggered" >;
ALLKW:          AllLex<"ALL" >
                |<"all" >;
SOMEKW:         SomeLex<"SOME" >
                |<"some" >;
PERKW:          PeriodLex<"PERIOD" >
                |<"period" >;
FINISHKW:       FinishLex<"FINISH" >
                |<"finish" >;
MCPKW:          MCPLex<"MINIMUM\ CALLING\ PERIOD" >
                |<"minimum\ call\ period" >;
MRTKW:          MRTLex<"MAXIMUM\ RESPONSE\ TIME" >
                |<"minimum\ response\ time" >;
MAXEXTIMEKW:    MaxExTimeLex<"MAXIMUM\ EXECUTION\ TIME" >
                |<"maximum\ execution\ time" >;
IFKW:           IfLex<"IF" >
                |<"if" >;
RESETKW:        RESETLex<"RESET\ TIMER" >
      |<"reset\ timer" >;
STARTKW:        STARTlex<"START\ TIMER" >
      |<"start\ timer" >;
```

APPENDIX B - Abstract Rules

```
STOPKW:          STOPLex<"STOP\ TIMER" >
      |<"stop\ timer" >;
QUOTEKW:     QUOTELex<"\"" >;
IDENTIFIER       : IdentLex<[a-zA-Z][a-zA-Z_0-9]* >;
INTEGERS         : IntegerLex<[0-9]+ >;

/*  STRING_LITERAL: StringLitLex< [^\"]+ >;  */

/*********************************************************************/
/* abstract grammar for comment lines to allow for free textual input
*/

list commentLines;
commentLines
  : exported CommentLinesNil()
  | exported CommentLinesPair(commentLine commentLines)
  ;

commentLine
  : exported CommentLineNil()
       ;

/* input syntax */
yCommentLine(synthesized commentLine a; );
yCommentLines(synthesized commentLines a; );

/* commentLine~ yCommentLine.a ; */
commentLines~ <NO_WHITESPACE> yCommentLines.a ;

yCommentLine
  ::=()           { $$.a = CommentLine(""); }
  | (CLINE)       { $$.a = Remove_Leading_Blanks_From_String(CLINE); }
  ;

yCommentLines
  ::=(yCommentLine)
    { $$.a = CommentLinesPair(yCommentLine.a, CommentLinesNil); }
  | (yCommentLine CLINEBREAK yCommentLines)
    { $$.a = CommentLinesPair(yCommentLine.a, yCommentLines$2.a); }
  ;


/*              IF OPTIONAL COMMENTS ARE EVER WANTED IN PSDL, THE FOLLOWING
                DECLARATIONS SHOULD COME HANDY.  REFFER TO REPS' AND
                TEITLEBOUND'S BOOK, CHAPTER EIGHT FOR AN EXPLANATION OF
                HOW THIS WORKS.
*/

/*****************************

optional optionalComment;
optionalComment
    : OptionalCommentNil()
        | OptionalCommentPrompt()
        | OptionalComment(commentLines)
        ;

yOptionalComment        (synthesized optionalComment a;);
yOptionalComment
        ::=()   ($$.a=OptionalCommentNil;)
        |  (LCURLY yCommentLines RCURLY){
                yCommentLines.tail=CommentLinesNil;
                $$.a=OptionalComment(yCommentLines.reversed);
                }
        ;
optionalComment ~ yOptionalComment.a;
**********************************/


/********  End of grammar for format free textual input ***************/
/*************************************************************************/
/*****          PSDL PROPPER BEGINS RIGHT HERE                    *******
******
*************************************************************************/
prototype: exported Prot(psdl_components);

list psdl_components;

psdl_components
        : exported PsdlNil()
        | exported PsdlPair(component psdl_components)
        ;

component
        : NoComponent()
        | Data(id type_spec type_impl)
        | exported Op(id operator_spec operator_impl)
        ;

id      : exported IdNull()
        | exported Id(IDENTIFIER)
        ;

integer
        : exported IntegerNull()
        | exported IntegerVal(INTEGERS)
        ;

/*
string_lit
  : exported StringNull()
  | exported StringLit(STRING_LITERAL)
  ;
*/


type_spec
        : TypeSpec(o_generic_params
                   o_type_decls
                   o_operators
                   o_keywords
                   o_informal_descs
                   o_formal_descs)
        ;

optional o_generic_params;
o_generic_params
        : GenericNone()
        | GenericPrompt()
        | Generic(type_declarations)
        ;

list type_declarations;
type_declarations
        : exported TypeDeclNil()
        | exported TypeDeclPair(a_decl type_declarations)
```

```
        ;

a_decl  : exported ADeclNil()
        | exported ADecl(id_list decl_type_name)
        ;

type_name
        : TypeNameNull()
        | TypeName(id o_bracket_type_declarations)
        ;

decl_type_name
        : exported DTypeNameNull()
        | exported DTypeInteger()
        | exported DTypeReal()
        | exported DTypeBoolean()
/*
  | exported DTypeException()
*/
        | exported DTypeSimpleId(id)
        | exported DTypeUserDefined(id bracket_type_declarations)
        ;

optional o_bracket_type_declarations;
o_bracket_type_declarations
        : OBTypeNone()
        | OBTypePrompt()
        | OBTypeDeclaration(type_declarations)
        ;

bracket_type_declarations
       : BTypeNull()
       | BTypeDeclaration(type_declarations)
       ;

list id_list;
id_list
        : exported IdNil()
        | exported IdPair(id id_list)
        ;

list alone_id_list;
alone_id_list
        : exported AIdNil()
        | exported AIdPair(id alone_id_list)
        ;

optional o_type_decls;
o_type_decls
        : TypeNone()
        | TypePrompt()
        | Type(type_declarations)
        ;

optional list o_operators;
o_operators
        : OperatorNil()
        | OperatorPair(t_oper_spec o_operators)
        ;
t_oper_spec
        : TOpSpecNil()
        | TOpSpec(id operator_spec)
  ;
```

```
operator_spec
        : exported OperatorSpec
                        (o_generics_list
                         o_inputs_list
                         o_outputs_list
                         o_states_list
                         o_exceptions_list
                         o_timing_info
                         o_keywords
                         o_informal_descs
                         o_formal_descs)
        ;

optional list o_generics_list;
o_generics_list
        : GenericsListNone()
        | GenericsListPair(o_generics
                         o_generics_list)
        ;

o_generics
        : OpGenericsNone()
        | OpGenerics(type_declarations reqmts_trace)
        ;

optional list o_inputs_list;
o_inputs_list
        : exported InputsListNone()
        | exported InputsListPair(o_inputs
                                        o_inputs_list)
        ;

o_inputs
        : exported OpInputsNone()
        | exported OpInputs(type_declarations reqmts_trace)
        ;

optional list o_outputs_list;
o_outputs_list
        : OutputsListNone()
        | OutputsListPair(o_outputs
                          o_outputs_list)
        ;

o_outputs
        : OpOutputsNone()
        | OpOutputs(type_declarations reqmts_trace)
        ;

optional list o_exceptions_list;
o_exceptions_list
        : ExcListNone()
        | ExcListPair(o_exceptions
                      o_exceptions_list)
        ;

o_exceptions
        : OpExceptionsNone()
        | OpExceptions(alone_id_list reqmts_trace)
        ;

/*
```

```
optional list o_timing_info_list;
o_timing_info_list
        : TimeInfoNone()
        | TimeInfoPair(o_timing_info
                        o_timing_info_list)
        ;
*/

optional o_timing_info;
o_timing_info
        : exported OpTimingInfoNone()
        | exported OpTimingInfoPrompt()
        | exported OpTimingInfo(time reqmts_trace)
        ;

time
        : exported TimeNull()
        | exported Time(integer time_unit);

time_unit
        : exported UnitNil()
        | exported UnitMICROSECONDS()
        | exported UnitMS()
        | exported UnitSEC()
        | exported UnitMIN()
        | exported UnitHOURS()
        ;

optional list o_states_list;
o_states_list
        : exported StatesListNone()
        | exported StatesListPair(o_states
                                                o_states_list);

o_states
        : exported OpStatesNone()
        | exported OpStates(type_declarations expression_list reqmts_trace)
        ;

optional list initial_args;
/*
optional initial_args;
*/
initial_args
        : InitialArgsNil()
/*
        | InitialArgsPrompt()
        | InitialArgs(expression_list)
*/
        | InitialArgs(an_argument initial_args)
        ;

an_argument
        : AnArgNil()
        | AnArgument(expression_list)
        ;

list expression_list;
expression_list
        : InitialExpListNil()
        | InitialExpListPair(expression expression_list)
        ;
```

```
expression
        : ExpNull()
        | Identifier(id)
        | Textual_Description(commentLines)
/*
        | Textual_Description(string_lit)
*/
        | TypeExpression(type_name id initial_args)
        | ParenthesizedExp(expression)

  /*  BOOLEAN EXPRESSIONS  */

        | True()
        | False()
        | NotExp(expression)
        | EqualExp(expression expression)
        | LessExp(expression expression)
        | GreaterExp(expression expression)
        | GreatEqualExp(expression expression)
        | LessEqualExp(expression expression)
        | NotEquivExp(expression expression)
        | AndExp(expression expression)
        | OrExp(expression expression)
        | XorExp(expression expression)

  /* ARITHMETIC EXPRESSIONS */

        | Integer(integer)
        | Real(integer integer)
        | PlusExp(expression expression)
        | MinusExp(expression expression)
        | TimesExp(expression expression)
        | DivExp(expression expression)
        | NegativeExp(expression)
  | PositiveExp(expression)
        | AbsExp(expression)
        | RemExp(expression expression)
        | ModExp(expression expression)
        | ExponentExp(expression expression)

  /* STRING EXPRESSION */
  | ConcatExp(expression expression)
        ;

optional o_keywords;
o_keywords
        : KeyWordsNone()
        | KeyWordsPrompt()
        | KeyWords(alone_id_list)
        ;

optional o_informal_descs;
o_informal_descs
        : exported InformalDescsNull()
        | exported InformalDescsPrompt()
        | exported InformalDescs(commentLines)
        ;

optional o_formal_descs;
o_formal_descs
        : FormalDescsNone()
        | FormalDescsPrompt()
        | FormalDescs(commentLines)
```

```
        ;

optional reqmts_trace;
reqmts_trace
        : ReqmtsTraceNone()
        | ReqmtsTracePrompt()
        | ReqmtsTrace(alone_id_list)
        ;


text    : TextNull()
        | Text(id)
        ;

type_impl
        : TypeImplNull()
        | AdaTypeImpl(id)
        | TypeImpl(type_name
                   operator_impl_list)
        ;

/*      new declarations    */

optional list operator_impl_list;
operator_impl_list
        : OpImplListNull()
        | OpImplListPair(t_op_impl
                         operator_impl_list)
        ;

t_op_impl
        : TOpImplNull()
        | TOpImpl(id
                  operator_impl)
        ;

operator_impl
        : exported OpImplNull()
        | exported AdaOpImpl(id)
        | exported OperatorImpl(graph
                                declarations
                                cc)
        ;

graph
        : exported GraphNull()
        | exported Graph(vertex_list
                                     edge_list)
        ;

optional list vertex_list;
vertex_list
        : exported VertexListNull()
        | exported VertexListPair(a_vertex
                                                        vertex_list)
        ;

a_vertex
        : exported AVertexNull()
        | exported AVertex(operator_id
                                       optional_time)
        ;
```

```
operator_id
        : exported OperatorIdNull()
        | exported OperatorId(optional_type_id
                              id
                              operator_id_pairs)
        ;

optional optional_type_id;
optional_type_id
  : exported OptionalTypeIdNull()
  | exported OptionalTypeIdPrompt()
  | exported OptionalTypeId(id)
  ;

optional operator_id_pairs;
operator_id_pairs
        : exported OperatorIdPairsNull()
        | exported OperatorIdPairsPrompt()
        | exported OperatorIdPairs(alone_id_list
                                                        alone_id_list)
        ;

optional optional_time;
optional_time
        : exported OptionalTimeNull()
        | exported OptionalTimePrompt()
        | exported OptionalTime(time)
        ;

optional list edge_list;
edge_list
        : exported EdgeListNil()
        | exported EdgeListPair(an_edge
                                              edge_list)
        ;

an_edge
        : exported AnEdgeNull()
        | exported AnEdge(id
                                    latency_time
                                    from_vertex_id
                                    to_vertex_id)
        ;

optional latency_time;
latency_time
        : exported LatencyTimeNull()
        | exported LatencyTimePrompt()
        | exported LatencyTime(time);

from_vertex_id
        : FVertexIdNull()
        | FVertexId (optional_type_id
                id
                      operator_id_pairs)
        ;

to_vertex_id
        : TVertexIdNull()
        | TVertexId(optional_type_id
                id
               operator_id_pairs)
        ;
```

APPENDIX B - Abstract Rules

```
declarations
/*
        : exported DeclarationsNull()
        | exported Declarations(optional_streams
*/
        : exported Declarations(optional_streams
                                        optional_timers)
        ;

optional optional_streams;
optional_streams
        : exported StreamsNull()
        | exported StreamsPrompt()
        | exported Streams(type_declarations)
        ;

optional optional_timers;
optional_timers
        : exported TimersNull()
        | exported TimersPrompt()
        | exported Timers(alone_id_list)
        ;

cc      : exported CcNull()
        | exported Cc(constraints
            o_informal_descs)
        ;

list constraints;
constraints
        : exported ConstraintsNull()
        | exported ConstraintsPair(a_constraint
                        constraints)
        ;

a_constraint
        : exported AConstraintNull()
        | exported AConstraint(operator_id
                    optional_trigger
                    optional_period
                    optional_finish_within
                    optional_mcp
                    optional_mrt
                    output_guards
                    exception_ops
                    timer_operations)
        ;

optional optional_trigger;
optional_trigger
        : exported OptionalTriggerNull()
        | exported OptionalTriggerPrompt()
        | exported OptionalTriggerAllOrSome(
                            type_of_trigger
                            alone_id_list
                            optional_if_predicate
                            reqmts_trace)

        | exported OptionalIfExp(expression
                        reqmts_trace)
        ;
```

```
type_of_trigger
        : exported TriggerNull()
        | exported TriggerAll()
        | exported TriggerSome()
        ;

optional optional_period;
optional_period
        : exported OptPeriodNull()
        | exported OptPeriodPrompt()
        | exported OptPeriod(time
                    reqmts_trace)
        ;

optional optional_finish_within;
optional_finish_within
        : exported OptFinishWithinNull()
        | exported OptFinishWithinPrompt()
        | exported OptFinishWithin(time
                        reqmts_trace)
        ;

optional optional_mcp;
optional_mcp
        : exported OptMcpNull()
        | exported OptMcpPrompt()
        | exported OptMcp(time reqmts_trace)
        ;

optional optional_mrt;
optional_mrt
        : exported OptMrtNull()
        | exported OptMrtPrompt()
        | exported OptMrt(time reqmts_trace)
        ;

optional list output_guards;
output_guards
        : exported OutputGuardsNil()
        | exported OutputGuardsPair(a_guard output_guards)
        ;

a_guard : exported AGuardNull()
        | exported AGuard(alone_id_list
                c_expression
                reqmts_trace)
        ;

optional exception_ops;
exception_ops
        : exported ExceptionOpsNull()
        | exported ExceptionOpsPrompt()
        | exported Exception(exception_options)
        ;

optional list exception_options;
exception_options
        : exported ExceptionOptionsNil()
        | exported ExceptionOptionsPair(an_exception
                                exception_options)
        ;

an_exception
```

```
        : exported AnExceptionNull()
        | exported AnException(id
                    optional_if_predicate
                    reqmts_trace)
        ;

optional list timer_operations;
timer_operations
        : exported TimerOperationsNil()
        | exported TimerOperationsPair(a_timer_operation
                        timer_operations)
        ;

a_timer_operation
        : exported ATimerOperationNull()
        | exported ATimerReset(id optional_if_predicate reqmts_trace)
        | exported ATimerStop(id optional_if_predicate reqmts_trace)
        | exported ATimerStart(id optional_if_predicate reqmts_trace)
;

optional optional_if_predicate;
optional_if_predicate
        : OptIfPredicateNull()
        | OptIfPredicatePrompt()
        | OptIfPredicate(c_expression)
        ;

/*
if_expression
        : IfExpressionNull()
        | IfExpression(c_expression)
        ;
*/


/*  EXPRESSIONS FOR IF'S    */

optional list c_initial_args;
/*
optional c_initial_args;
*/
c_initial_args
        : CInitialArgsNil()
/*
    | CInitialArgsPrompt()
        | CInitialArgs(c_expression_list)
*/
        | CInitialArgs(c_an_argument c_initial_args)
        ;

c_an_argument
        : CAnArgNil()
        | CAnArgument(c_expression_list)
        ;

list c_expression_list;
c_expression_list
        : CInitialExpListNil()
        | CInitialExpListPair(c_expression c_expression_list)
        ;

c_expression
        : CExpNull()
        | CIdentifier(id)
        | CTextual_Description(commentLines)
/*
        | CTextual_Description(string_lit)
*/
        | CTypeExpression(type_name id c_initial_args)
        | CTimeExpression(time)
        | CParenthesizedExp(c_expression)

  /* BOOLEAN EXPRESSIONS */

        | CTrue()
        | CFalse()
        | CNotExp(c_expression)
        | CEqualExp(c_expression c_expression)
        | CLessExp(c_expression c_expression)
        | CGreaterExp(c_expression c_expression)
        | CGreatEqualExp(c_expression c_expression)
        | CLessEqualExp(c_expression c_expression)
        | CNotEquivExp(c_expression c_expression)
        | CAndExp(c_expression c_expression)
        | COrExp(c_expression c_expression)
        | CXorExp(c_expression c_expression)

  /* INTEGER EXPRESSIONS */

        | CInteger(integer)
        | CReal(integer integer)
        | CPlusExp(c_expression c_expression)
        | CMinusExp(c_expression c_expression)
        | CTimesExp(c_expression c_expression)
        | CDivExp(c_expression c_expression)
        | CNegativeExp(c_expression)
        | CPositiveExp(c_expression)
        | CAbsExp(c_expression)
        | CRemExp(c_expression c_expression)
        | CModExp(c_expression c_expression)
        | CExponentExp(c_expression c_expression)

  /* STRING EXPRESSIONS */

        | CConcatExp(c_expression c_expression)
        ;
/*****************************************************************
****  Below are Abstract Rules added by
****  Scott Grosenheider
****  on March 1996
****  for work that implements semantic checking of timing and
****  control constraints within the SDE.
*****************************************************************/

list op_id_met_set;

op_id_met_set
   : exported OpIdMetSetNil()
   | exported OpIdMetPair(op_id_met op_id_met_set)
   ;


op_id_met
   : exported OpIdMetNull()
   | exported OpIdMet(operator_id met)
   ;
```

```
/* Maximum execution time */
/* Due to problems when working w/ time, all constraint times will be
   converted to a common integer (assume microseconds)              */
met
   : exported MetNull()
   | exported MET(REAL)
   ;

/*-----------------------------------------------------*/

list id_constraint_set;

id_constraint_set
   : exported IdConstraintSetNil()
   | exported IdConstraintPair(id_constraint id_constraint_set)
   ;

id_constraint
   : exported IdConstraintNull()
   | exported IdConstraint(operator_id time_constraints trigger)
   ;

time_constraints
   : exported OpConstraintsNull()
   | exported Periodic(per fw)
   | exported Sporadic(mcp mrt)
   ;

trigger
   : exported TriggerByNull()
   | exported TriggerByAll()
   | exported TriggerBySome()
   ;
/* Period */
per
   : exported PerNull()
   | exported Per(REAL)
   ;

/* Finish Within */
fw
   : exported FwNull()
   | exported FW(REAL)
   ;

/* Minimum calling period */
mcp
   : exported McpNull()
   | exported MCP(REAL)
   ;

/* Maximun response time */
mrt
   : exported MrtNull()
   | exported MRT(REAL)
   ;


/*-----------------------------------------------------*/
list edge_set;

edge_set
   : exported EdgeSetNil()
   | exported EdgePair(edge edge_set)
   ;

edge
   : exported EdgeNull()
   | exported Edge(id latency producer consumer)
   ;

latency
   : exported LatencyNull()
   | exported Latency(REAL)
   ;

producer
   : exported ProducerNull()
   | exported Producer(operator_id)
   ;

consumer
   : exported ConsumerNull()
   | exported Consumer(operator_id)
   ;
```

APPENDIX C - Attribute Rules

```
prototype, psdl_components
    (
     syn psdl_components syn_defined_operators;
     syn psdl_components syn_defined_types;

/*
     syn type_declarations syn_defined_streams;
     syn type_declarations syn_defined_states;
*/

     syn type_declarations syn_defined_type_decl;
     syn vertex_list syn_defined_vertices;
/*
     syn edge_list syn_defined_edges;
*/
    );

/*-------------------------------------------------------------------------*/

prototype, psdl_components, component (syn op_id_met_set     syn_vertex_id_met_set;);
prototype, psdl_components, component (syn id_constraint_set syn_id_constraint_set;);

prototype
    (syn IdSet syn_root_ids;
     syn IdSet syn_multiple_root_ids;
     syn IdSet syn_multiple_op_spec;
     syn IdSet syn_multiple_type_spec;
     syn OpIdSet syn_multiple_vertices;
     syn IdSet syn_multiple_streams;
    );


prototype
  : Prot (
      $$.syn_vertex_id_met_set = psdl_components.syn_vertex_id_met_set;
      $$.syn_id_constraint_set = psdl_components.syn_id_constraint_set;
    $$.syn_defined_operators = psdl_components.syn_defined_operators;
    $$.syn_defined_types = psdl_components.syn_defined_types;


    $$.syn_defined_type_decl = psdl_components.syn_defined_type_decl;

    $$.syn_defined_vertices = psdl_components.syn_defined_vertices;

    local IdSet root_components;
    root_components = Extract_Root_Components(
          psdl_components.syn_defined_operators,
          psdl_components.syn_defined_vertices);
    $$.syn_root_ids = root_components;

    local BOOL multiple_root_error;
    multiple_root_error = (IdSetSize(root_components) > 1);

    local STR multiple_root_message;
    multiple_root_message = (multiple_root_error)
          ? "\n--\n-- Multiple Root Operators \n"
          : "";

    local IdSet multiple_root_ids;
    multiple_root_ids = (multiple_root_error)
                              ? root_components
                              : IdSetNil;
    $$.syn_multiple_root_ids = multiple_root_ids;

    local IdSet multiple_op_spec;
    multiple_op_spec = Extract_Multiple_Op_Spec_Id(
          psdl_components.syn_defined_operators);
    $$.syn_multiple_op_spec = multiple_op_spec;

    local BOOL multiple_op_spec_error;
    multiple_op_spec_error = (!IsNull(multiple_op_spec));

    local STR multiple_op_spec_message;
    multiple_op_spec_message = (multiple_op_spec_error)
        ? "\n-- \n-- Components With More Than 1 Operator Spec\n"
        : "";

    local IdSet multiple_type_spec;
    multiple_type_spec = Extract_Multiple_Type_Spec_Id(
          psdl_components.syn_defined_types);
    $$.syn_multiple_type_spec = multiple_type_spec;

    local BOOL multiple_type_spec_error;
    multiple_type_spec_error = (!IsNull(multiple_type_spec));

    local STR multiple_type_spec_message;
    multiple_type_spec_message = (multiple_type_spec_error)
        ? "\n-- \n-- Components With More Than 1 Type Spec\n"
        : "";
    local IdSet also_op_type_ids;
    also_op_type_ids = Extract_Op_Type_Spec_Id(
          psdl_components.syn_defined_types,
          psdl_components.syn_defined_operators);

    local BOOL also_op_type_error;
    also_op_type_error = (!IsNull(also_op_type_ids));

    local STR also_op_type_message;
    also_op_type_message = (also_op_type_error)
        ? "\n-- \n-- Components Defined As Both Operators and Types\n"
        : "";

    local IdSet undefined_op_spec_set;
    undefined_op_spec_set = Extract_Undefined_Op(
            psdl_components.syn_defined_vertices,
            psdl_components.syn_defined_operators);

    local BOOL undefined_op_spec_error;
    undefined_op_spec_error = (!IsNull(undefined_op_spec_set));

    local STR undefined_op_spec_message;
    undefined_op_spec_message = (undefined_op_spec_error)
        ? "\n-- \n-- Operators Missing Operator Spec\n"
        : "";

    local OpIdSet undefined_type_op_spec_set;
    undefined_type_op_spec_set = Extract_Undefined_Type_Op(
            psdl_components.syn_defined_vertices,
            psdl_components.syn_defined_types);

    local BOOL undefined_type_op_spec_error;
    undefined_type_op_spec_error = (!OpIdSetIsNull(undefined_type_op_spec_set));

    local STR undefined_type_op_spec_message;
    undefined_type_op_spec_message = (undefined_type_op_spec_error)
        ? "\n-- \n-- Operators Missing Type and Op Spec\n"
```

```
        : "";


    local OpIdSet multiple_vertices;
    multiple_vertices = Extract_Multiple_Vertices(
            psdl_components.syn_defined_vertices);
    $$.syn_multiple_vertices = multiple_vertices;


    local BOOL multiple_vertices_error;
    multiple_vertices_error = (!OpIdSetIsNull(multiple_vertices));

    local STR multiple_vertices_message;
    multiple_vertices_message = (multiple_vertices_error)
        ? "\n-- \n-- Vertices Appeared In More Than 1 Operator Impl\n"
        : "";

    local IdSet multiple_streams;
    multiple_streams = Extract_Multiple_TD(psdl_components.syn_defined_type_decl);
    $$.syn_multiple_streams = multiple_streams;

    local BOOL multiple_streams_error;
    multiple_streams_error = (!IsNull(multiple_streams));

    local STR multiple_streams_message;
    multiple_streams_message = (multiple_streams_error)
        ? "\n-- \n-- Streams With More Than 1 State/Data Stream Declaration\n"
        : "";

      local INT  processor_number;
      processor_number = Min_Processors_Required($$.syn_vertex_id_met_set,
                                                 $$.syn_id_constraint_set);
      local BOOL min_processor_required_GT1;
      min_processor_required_GT1 =
                  ( 1 < processor_number);
      local STR min_processor_msg;
      min_processor_msg =
            (min_processor_required_GT1)
            ? "\n-- !Prototype will not schedule for less than\n"
            # "-- "
            # INTtoSTR(processor_number)
            # " processors. (Based on Periodic Operators only)"
            : "";
/*--------------*/

      local BOOL uniprocessor_unschedulability_error;
      uniprocessor_unschedulability_error =
                      !min_processor_required_GT1 &&
                      !Uniprocessor_Schedulable($$.syn_vertex_id_met_set,
                                                $$.syn_id_constraint_set);
      local STR uniprocessor_unschedulability_msg;
      uniprocessor_unschedulability_msg =
           (uniprocessor_unschedulability_error)
           ? "\n-- !Prototype will not schedule for Uniprocessor.\n"
           # "--      (Based on Periodic Operators only)"
           : "";


    local BOOL has_error;
    has_error = (multiple_root_error
        || multiple_op_spec_error
        || multiple_type_spec_error
        || also_op_type_error
        || undefined_op_spec_error
        || undefined_type_op_spec_error
        || multiple_vertices_error
        || multiple_streams_error
           || min_processor_required_GT1
           || uniprocessor_unschedulability_error);


    local STR error_header;
    error_header = (has_error)
        ? "----------------------------------------- \n-- WARNINGS, ERRORS AND
ALERTS:"
        : "";


    local STR error_trailer;
    error_trailer = (has_error)
        ? "\n-- \n----------------------------------------- \n\n\n"
        : "";

    store(default_store global_proto_store) local prototype global_proto;
    global_proto = prototype;

    store(default_store global_root_store) local IdSet global_root;
    global_root = root_components;

    store(default_store global_type_decl_store) local type_declarations
global_type_decl;
    global_type_decl = psdl_components.syn_defined_type_decl;

    store(default_store global_undef_ops_store) local IdSet global_undef_ops;
    global_undef_ops = undefined_op_spec_set;

  };

/*-------------------------------------------------------------------------*/

psdl_components
  : PsdlNil
    ($$.syn_defined_operators = PsdlNil;
     $$.syn_defined_types = PsdlNil;
     $$.syn_defined_type_decl = TypeDeclNil;
     $$.syn_defined_vertices = VertexListNull;
       $$.syn_vertex_id_met_set = OpIdMetSetNil;
       $$.syn_id_constraint_set = IdConstraintSetNil;
    )
  | PsdlPair
    ($$.syn_defined_operators =
        (IsOperator(component)
            ? GetOperator(component)::
              psdl_components$2.syn_defined_operators
            : psdl_components$2.syn_defined_operators);

     $$.syn_defined_types =
        (IsType(component)
            ? GetType(component)::
              psdl_components$2.syn_defined_types
            : psdl_components$2.syn_defined_types);


    $$.syn_defined_type_decl =
        Concat_Type_Decl_List(
          Concat_Type_Decl_List(
```

```
            Concat_Type_Decl_List(
              Get_States(component),
              Get_Streams(component)),
            (IsType(component)
            ? Concat_Type_Decl_List(
              Get_Inputs(component),
              Get_Outputs(component))
            : TypeDeclNil)),
          psdl_components$2.syn_defined_type_decl);


    $$.syn_defined_vertices =
        Concat_Vertex_List(
          GetVertices(component),
          psdl_components$2.syn_defined_vertices);

     $$.syn_vertex_id_met_set =
           Op_Id_Met_Set_Union(component.syn_vertex_id_met_set,
                               psdl_components$2.syn_vertex_id_met_set);
     $$.syn_id_constraint_set =
           Id_Constraint_Set_Union(component.syn_id_constraint_set,
                                   psdl_components$2.syn_id_constraint_set);
  };

/*-------------------------------------------------------------------------------*/

type_declarations, a_decl, id_list {synthesized IdSet id_set; };

id_list {inh type_declarations inh_types_types;
     inh type_declarations inh_generic_types;
     inh type_declarations inh_generics_types;
     inh type_declarations inh_inputs_types;
     inh type_declarations inh_outputs_types;
     inh type_declarations inh_states_types;
  };

id_list
  : IdNil
    ($$.id_set = IdSetNil;
    )
  | IdPair
    ($$.id_set = IdSetUnion(SingletonIdSet(id), id_list$2.id_set);

     id_list$2.inh_types_types = $$.inh_types_types;
     id_list$2.inh_generic_types = $$.inh_generic_types;

     id_list$2.inh_generics_types = $$.inh_generics_types;
     id_list$2.inh_inputs_types = $$.inh_inputs_types;
     id_list$2.inh_outputs_types = $$.inh_outputs_types;
     id_list$2.inh_states_types = $$.inh_states_types;
    );

id_list:
  IdPair
    (local STR multiply_defined;
     multiply_defined =
        (((id != IdNull)&&
          ((IdIsDefInTypes(id, $$.inh_types_types) +
            IdIsDefInTypes(id, $$.inh_generic_types)+
            IdIsDefInTypes(id, $$.inh_generics_types)+
            IdIsDefInTypes(id, $$.inh_inputs_types)+
            IdIsDefInTypes(id, $$.inh_outputs_types)+
            IdIsDefInTypes(id, $$.inh_states_types)) > 1))
         ? " >--MULTIPLY DEFINED"
         : "");
  };

/*-------------------------------------------------------------------------------*/

type_declarations, a_decl
  (inh type_declarations inh_types_types;
   inh type_declarations inh_generic_types;

   inh type_declarations inh_generics_types;
   inh type_declarations inh_inputs_types;
   inh type_declarations inh_outputs_types;
   inh type_declarations inh_states_types;
  );

type_declarations
  : TypeDeclNil
    ($$.id_set = IdSetNil;
    )
  | TypeDeclPair
    ($$.id_set = IdSetUnion(a_decl.id_set, type_declarations$2.id_set);

     a_decl.inh_types_types = $$.inh_types_types;
     type_declarations$2.inh_types_types = $$.inh_types_types;

     a_decl.inh_generic_types = $$.inh_generic_types;
     type_declarations$2.inh_generic_types = $$.inh_generic_types;

     type_declarations$2.inh_generics_types = $$.inh_generics_types;
     type_declarations$2.inh_inputs_types = $$.inh_inputs_types;
     type_declarations$2.inh_outputs_types = $$.inh_outputs_types;
     type_declarations$2.inh_states_types = $$.inh_states_types;

     a_decl.inh_generics_types = $$.inh_generics_types;
     a_decl.inh_inputs_types = $$.inh_inputs_types;
     a_decl.inh_outputs_types = $$.inh_outputs_types;
     a_decl.inh_states_types = $$.inh_states_types;


     );

/*-------------------------------------------------------------------------------*/

a_decl
  : ADeclNil
    ($$.id_set = IdSetNil;
    )
  | ADecl
    ($$.id_set = id_list.id_set;

     id_list.inh_types_types = $$.inh_types_types;
     id_list.inh_generic_types = $$.inh_generic_types;
     id_list.inh_generics_types = $$.inh_generics_types;
     id_list.inh_inputs_types = $$.inh_inputs_types;
     id_list.inh_outputs_types = $$.inh_outputs_types;
     id_list.inh_states_types = $$.inh_states_types;
    );

a_decl:
  ADecl (local STR undefined_ADT;
       undefined_ADT =
      (((DefinedType(GetDeclTypeId(decl_type_name), (prototype.syn_defined_types)) ==
```

```
0)
        && (GetDeclTypeId(decl_type_name) != IdNull))
      ? " >--UNDEFINED ADT"
      : "");
  };

/*-------------------------------------------------------------------------*/

o_generic_params
  {syn type_declarations declared_types;
   inh type_declarations inh_types_types;
   inh type_declarations inh_generic_types;
  };

o_generic_params
  : GenericNone
    {o_generic_params.declared_types = TypeDeclNil;}
  | GenericPrompt
    {o_generic_params.declared_types = TypeDeclNil;}
  | Generic
    {o_generic_params.declared_types = type_declarations;

     type_declarations.inh_types_types = $$.inh_types_types;
     type_declarations.inh_generic_types = $$.inh_generic_types;
     type_declarations.inh_generics_types = TypeDeclNil;
     type_declarations.inh_inputs_types = TypeDeclNil;
     type_declarations.inh_outputs_types = TypeDeclNil;
     type_declarations.inh_states_types = TypeDeclNil;
    }
  ;
/*-------------------------------------------------------------------------*/

o_type_decls
  {syn type_declarations declared_types;
   inh type_declarations inh_types_types;
   inh type_declarations inh_generic_types;
  };

o_type_decls
  : TypeNone
      {o_type_decls.declared_types = TypeDeclNil;}
  | TypePrompt
      {o_type_decls.declared_types = TypeDeclNil;}
  | Type
    {o_type_decls.declared_types = type_declarations;

     type_declarations.inh_types_types = $$.inh_types_types;
     type_declarations.inh_generic_types = $$.inh_generic_types;
     type_declarations.inh_generics_types = TypeDeclNil;
     type_declarations.inh_inputs_types = TypeDeclNil;
     type_declarations.inh_outputs_types = TypeDeclNil;
     type_declarations.inh_states_types = TypeDeclNil;
  };

/*-------------------------------------------------------------------------*/

type_spec{syn type_declarations generic_types;
      syn type_declarations types_types;
     syn o_operators defined_operators;
  };

type_spec
  : TypeSpec{
    $$.generic_types = o_generic_params.declared_types;
    $$.types_types = o_type_decls.declared_types;
    $$.defined_operators = o_operators;

    o_generic_params.inh_types_types = $$.types_types;
    o_type_decls.inh_types_types = $$.types_types;
    o_generic_params.inh_generic_types =$$.generic_types;
    o_type_decls.inh_generic_types = $$.generic_types;

  };

/*-------------------------------------------------------------------------*/
type_impl{ syn operator_impl_list syn_operator_impl_list;
    };

type_impl
  : TypeImplNull{
      $$.syn_operator_impl_list = OpImplListNull;
    }
  | AdaTypeImpl{
      $$.syn_operator_impl_list = OpImplListNull;
    }
  | TypeImpl{
      $$.syn_operator_impl_list = operator_impl_list;
    };

/*-------------------------------------------------------------------------*/

operator_spec, o_inputs_list, o_inputs { syn IdSet input_id_set; };

operator_spec, o_outputs_list, o_outputs { synthesized IdSet output_id_set; };

operator_spec, o_states_list, o_states { synthesized IdSet state_id_set; };

operator_impl, graph, vertex_list, a_vertex, operator_id { synthesized OpIdSet
vertex_id_set; };
operator_impl, graph, vertex_list (syn op_id_met_set     syn_vertex_id_met_set;);
operator_impl, graph, edge_list   (syn edge_set          syn_edge_set;);
operator_impl, cc, constraints    (syn id_constraint_set syn_id_constraint_set;);
cc, constraints, a_constraint     (inh op_id_met_set     inh_vertex_id_met_set;);


operator_impl, graph, edge_list, an_edge { synthesized IdSet edge_id_set; };

operator_impl, declarations, optional_streams { synthesized IdSet stream_id_set; };

operator_impl, cc, constraints, a_constraint { synthesized OpIdSet
constraint_op_id_set; };


o_generics_list {syn type_declarations declared_types;};

o_generics_list, o_generics
  {
   inh type_declarations inh_generics_types;
   inh type_declarations inh_inputs_types;
   inh type_declarations inh_outputs_types;
   inh type_declarations inh_states_types;
  };

o_generics_list
  : GenericsListNone {$$.declared_types = TypeDeclNil;}
  | GenericsListPair
```

```
    ($$.declared_types = GetGenericsTypes(o_generics)@
                o_generics_list$2.declared_types;

     o_generics_list$2.inh_generics_types = $$.inh_generics_types;
     o_generics_list$2.inh_inputs_types = $$.inh_inputs_types;
     o_generics_list$2.inh_outputs_types = $$.inh_outputs_types;
     o_generics_list$2.inh_states_types = $$.inh_states_types;

     o_generics.inh_generics_types = $$.inh_generics_types;
     o_generics.inh_inputs_types = $$.inh_inputs_types;
     o_generics.inh_outputs_types = $$.inh_outputs_types;
     o_generics.inh_states_types = $$.inh_states_types;
  );

o_generics
  : OpGenerics
    (type_declarations.inh_types_types = TypeDeclNil;
     type_declarations.inh_generic_types = TypeDeclNil;

     type_declarations.inh_generics_types = $$.inh_generics_types;
     type_declarations.inh_inputs_types = $$.inh_inputs_types;
     type_declarations.inh_outputs_types = $$.inh_outputs_types;
     type_declarations.inh_states_types = $$.inh_states_types;
  );
/*-------------------------------------------------------------------------*/
o_inputs_list (syn type_declarations declared_types;);

o_inputs_list, o_inputs
  (
   inh type_declarations inh_generics_types;
   inh type_declarations inh_inputs_types;
   inh type_declarations inh_outputs_types;
   inh type_declarations inh_states_types;
  );

o_inputs_list
  : InputsListNone
    ($$.declared_types = TypeDeclNil;
     $$.input_id_set = IdSetNil;
    )
  | InputsListPair
    ($$.declared_types = GetInputsTypes(o_inputs)@ o_inputs_list$2.declared_types;
     $$.input_id_set = IdSetUnion(o_inputs.input_id_set, o_inputs_list$2.input_id_set);


     o_inputs_list$2.inh_generics_types = $$.inh_generics_types;
     o_inputs_list$2.inh_inputs_types = $$.inh_inputs_types;
     o_inputs_list$2.inh_outputs_types = TypeDeclNil;
     o_inputs_list$2.inh_states_types = $$.inh_states_types;

     o_inputs.inh_generics_types = $$.inh_generics_types;
     o_inputs.inh_inputs_types = $$.inh_inputs_types;
     o_inputs.inh_outputs_types = TypeDeclNil;
     o_inputs.inh_states_types = $$.inh_states_types;
  );

o_inputs
  : OpInputsNone
    ($$.input_id_set = IdSetNil;
    )
  | OpInputs
    ($$.input_id_set = type_declarations.id_set;

     type_declarations.inh_types_types = TypeDeclNil;
     type_declarations.inh_generic_types = TypeDeclNil;

     type_declarations.inh_generics_types = $$.inh_generics_types;
     type_declarations.inh_inputs_types = $$.inh_inputs_types;
     type_declarations.inh_outputs_types = TypeDeclNil;
     type_declarations.inh_states_types = $$.inh_states_types;
  );
/*-------------------------------------------------------------------------*/
o_outputs_list (syn type_declarations declared_types;);

o_outputs_list, o_outputs
  (
   inh type_declarations inh_generics_types;
   inh type_declarations inh_inputs_types;
   inh type_declarations inh_outputs_types;
   inh type_declarations inh_states_types;
  );

o_outputs_list
  : OutputsListNone
    ($$.declared_types = TypeDeclNil;
     $$.output_id_set = IdSetNil;
    )

  | OutputsListPair
    ($$.declared_types = GetOutputsTypes(o_outputs)@ o_outputs_list$2.declared_types;
     $$.output_id_set = IdSetUnion(o_outputs.output_id_set,
o_outputs_list$2.output_id_set);

     o_outputs_list$2.inh_generics_types = $$.inh_generics_types;
     o_outputs_list$2.inh_inputs_types = TypeDeclNil;
     o_outputs_list$2.inh_outputs_types = $$.inh_outputs_types;
     o_outputs_list$2.inh_states_types = $$.inh_states_types;

     o_outputs.inh_generics_types = $$.inh_generics_types;
     o_outputs.inh_inputs_types = TypeDeclNil;
     o_outputs.inh_outputs_types = $$.inh_outputs_types;
     o_outputs.inh_states_types = $$.inh_states_types;
  );

o_outputs
  : OpOutputsNone
    ($$.output_id_set = IdSetNil;
    )
  | OpOutputs
    ($$.output_id_set = type_declarations.id_set;

     type_declarations.inh_types_types = TypeDeclNil;
     type_declarations.inh_generic_types = TypeDeclNil;

     type_declarations.inh_generics_types = $$.inh_generics_types;
     type_declarations.inh_inputs_types = TypeDeclNil;
     type_declarations.inh_outputs_types = $$.inh_outputs_types;
     type_declarations.inh_states_types = $$.inh_states_types;
  );

/*-------------------------------------------------------------------------*/
o_states_list (syn type_declarations declared_types;);

o_states_list, o_states
  (
   inh type_declarations inh_generics_types;
```

```
   inh type_declarations inh_inputs_types;
   inh type_declarations inh_outputs_types;
   inh type_declarations inh_states_types;
  );

o_states_list
  : StatesListNone
    {$$.declared_types = TypeDeclNil;
     $$.state_id_set = IdSetNil;
    }

  | StatesListPair
    {$$.declared_types = GetStatesTypes(o_states)@ o_states_list$2.declared_types;
     $$.state_id_set = IdSetUnion(o_states.state_id_set, o_states_list$2.state_id_set);

     o_states_list$2.inh_generics_types = $$.inh_generics_types;
     o_states_list$2.inh_inputs_types = $$.inh_inputs_types;
     o_states_list$2.inh_outputs_types = $$.inh_outputs_types;
     o_states_list$2.inh_states_types = $$.inh_states_types;

     o_states.inh_generics_types = $$.inh_generics_types;
     o_states.inh_inputs_types = $$.inh_inputs_types;
     o_states.inh_outputs_types = $$.inh_outputs_types;
     o_states.inh_states_types = $$.inh_states_types;
  );

o_states
  : OpStatesNone
    {$$.state_id_set = IdSetNil;
    }
  | OpStates
    {$$.state_id_set = type_declarations.id_set;

     type_declarations.inh_types_types = TypeDeclNil;
     type_declarations.inh_generic_types = TypeDeclNil;

     type_declarations.inh_generics_types = $$.inh_generics_types;
     type_declarations.inh_inputs_types = $$.inh_inputs_types;
     type_declarations.inh_outputs_types = $$.inh_outputs_types;
     type_declarations.inh_states_types = $$.inh_states_types;
  );
/*-------------------------------------------------------------------------------*/
o_timing_info
  (syn time syn_met;
  );

o_timing_info
  : OpTimingInfoNone( $$.syn_met = TimeNull; )
  | OpTimingInfoPrompt( $$.syn_met = TimeNull; )
  | OpTimingInfo( $$.syn_met = time; );

/*-------------------------------------------------------------------------------*/
operator_spec
  (syn type_declarations generics_types;
   syn type_declarations inputs_types;
   syn type_declarations outputs_types;
   syn type_declarations states_types;
   syn time syn_met;
  );

operator_spec
  : OperatorSpec(
    $$.generics_types = o_generics_list.declared_types;
    $$.inputs_types = o_inputs_list.declared_types;
    $$.outputs_types = o_outputs_list.declared_types;
    $$.states_types = o_states_list.declared_types;

    $$.input_id_set = o_inputs_list.input_id_set;
    $$.output_id_set = o_outputs_list.output_id_set;
    $$.state_id_set = o_states_list.state_id_set;

    o_generics_list.inh_generics_types = $$.generics_types;
    o_generics_list.inh_inputs_types = $$.inputs_types;
    o_generics_list.inh_outputs_types = $$.outputs_types;
    o_generics_list.inh_states_types = $$.states_types;

    o_inputs_list.inh_generics_types = $$.generics_types;
    o_inputs_list.inh_inputs_types = $$.inputs_types;
    o_inputs_list.inh_outputs_types = $$.outputs_types;
    o_inputs_list.inh_states_types = $$.states_types;

    o_outputs_list.inh_generics_types = $$.generics_types;
    o_outputs_list.inh_inputs_types = $$.inputs_types;
    o_outputs_list.inh_outputs_types = $$.outputs_types;
    o_outputs_list.inh_states_types = $$.states_types;

    o_states_list.inh_generics_types = $$.generics_types;
    o_states_list.inh_inputs_types = $$.inputs_types;
    o_states_list.inh_outputs_types = $$.outputs_types;
    o_states_list.inh_states_types = $$.states_types;

    $$.syn_met = o_timing_info.syn_met;
  );

/*-------------------------------------------------------------------------------*/
t_oper_spec
  : TOpSpec (

    local BOOL multiply_defined_error;
    multiply_defined_error = (id!=IdNull) && (OpIsDefinedinTypeSpec(
            id, (type_spec.defined_operators)) > 1);

    local STR multiply_defined_message;
          multiply_defined_message = (multiply_defined_error)
        ? " \n-- \n-- Multiply Declared Operator Spec"
        :"";

    local BOOL has_error;
    has_error = (multiply_defined_error );

    local STR error_header;
    error_header = (has_error)
        ? "\n-----------------------------------------\n-- WARNINGS, ERRORS AND
ALERTS:"
        : "";

    local STR error_trailer;
    error_trailer = (has_error)
        ? "\n-----------------------------------------\n"
        : "";

     store(default_store t_multi_op_error_store) local BOOL local_multi_op_error;
     local_multi_op_error = multiply_defined_error;

  );
```

```
/*-------------------------------------------------------------------------*/

t_op_impl
  : TOpImpl (
    local t_oper_spec local_t_oper_spec;
    local_t_oper_spec = (id!=IdNull)
        ? Find_T_Op_Spec(id, (Data.syn_o_operators))
        : TOpSpecNil;

    local BOOL missing_op_spec_error;
    missing_op_spec_error = (id!=IdNull)
        && (local_t_oper_spec == TOpSpecNil);

    local STR missing_op_spec_message;
         missing_op_spec_message = (missing_op_spec_error)
        ? " \n-- \n-- Obsolete Operator Implementation"
        :"";

    local BOOL multiply_defined_error;
               multiply_defined_error = ((id!=IdNull) &&
          (OpIsDefinedinTypeImpl(id,
            (type_impl.syn_operator_impl_list)) > 1));

    local STR multiply_defined_message;
         multiply_defined_message = (multiply_defined_error)
        ? " \n-- \n-- Multiply Declared Operator Impl"
        :"";

    local IdSet inh_input_id_set;
    inh_input_id_set = Get_Input_Id_Set_From_TOpSpec(local_t_oper_spec);

    local IdSet inh_output_id_set;
    inh_output_id_set = Get_Output_Id_Set_From_TOpSpec(local_t_oper_spec);

    local IdSet inh_state_id_set;
    inh_state_id_set = Get_State_Id_Set_From_TOpSpec(local_t_oper_spec);

               local time inh_met;
               inh_met = Get_Met_From_TOpSpec(local_t_oper_spec);

    local IdSet input_output_state_union_set;
    input_output_state_union_set =
        IdSetUnion(inh_state_id_set,
          IdSetUnion(inh_input_id_set, inh_output_id_set));

    local IdSet obsolete_state_id_set;
    obsolete_state_id_set = (Get_Impl_Form(operator_impl) == 2)
        ? IdSetDifference(
          inh_state_id_set,
          Extract_Edge_Id_Set(operator_impl))
        : IdSetNil;

    local BOOL obsolete_state_error;
               obsolete_state_error = (!IsNull(obsolete_state_id_set));

    local STR obsolete_state_message;
               obsolete_state_message = (obsolete_state_error)
                              ? "\n-- \n-- Obsolete State Declarations \n"
                              : "";


    local IdSet undefined_stream;
    undefined_stream =
          IdSetDifference(operator_impl.edge_id_set,
            IdSetUnion(operator_impl.stream_id_set,
              input_output_state_union_set));

    local IdSet obsolete_stream;
    obsolete_stream =
        IdSetUnion(
          IdSetDifference(operator_impl.stream_id_set,
            operator_impl.edge_id_set),
          IdSetIntersect(operator_impl.stream_id_set,
            input_output_state_union_set));

               local BOOL undefined_stream_error;
               undefined_stream_error = (!IsNull(undefined_stream));

               local STR undefined_stream_message;
               undefined_stream_message = (undefined_stream_error)
                              ? "\n-- \n-- Missing Stream Declarations \n"
                              : "";

               local BOOL obsolete_stream_error;
               obsolete_stream_error = (!IsNull(obsolete_stream));

               local STR obsolete_stream_message;
               obsolete_stream_message = (obsolete_stream_error)
                              ? "\n-- \n-- Obsolete Stream Declarations \n"
        : "";

               local id loc_operator_id;
                      loc_operator_id = id;

               local OpIdSet undefined_constraint;
               undefined_constraint = OpIdSetDifference(operator_impl.vertex_id_set,
            operator_impl.constraint_op_id_set);

               local OpIdSet obsolete_constraint;
               obsolete_constraint =
OpIdSetDifference(operator_impl.constraint_op_id_set,
            operator_impl.vertex_id_set);

               local BOOL undefined_constraint_error;
               undefined_constraint_error = (!OpIdSetIsNull(undefined_constraint));

               local STR undefined_constraint_message;
               undefined_constraint_message = (undefined_constraint_error)
                              ? "\n-- \n-- Missing Constraint Entries \n"
                              : "";

               local BOOL obsolete_constraint_error;
               obsolete_constraint_error = (!OpIdSetIsNull(obsolete_constraint));

               local STR obsolete_constraint_message;
               obsolete_constraint_message = (obsolete_constraint_error)

                              ? "\n-- \n-- Obsolete Constraint Entries \n"
                              : "";

               local BOOL has_error;
               has_error = (missing_op_spec_error ||
          multiply_defined_error ||
          obsolete_state_error ||
                              undefined_stream_error ||
                              obsolete_stream_error ||
```

```
                              undefined_constraint_error ||
                              obsolete_constraint_error);

              local STR error_header;
              error_header = (has_error)
                              ? "\n-------------------------------------------\n--
WARNINGS, ERRORS AND ALERTS:"

                              : "";


              local STR error_trailer;
              error_trailer = (has_error)
                              ? "\n-- \n-------------------------------------------
\n"
                              : "";


     store(default_store t_operator_id_store) local id operator_name;
     operator_name = id;

     store(default_store t_type_id_store) local id type_name;
     type_name = (Data.local_type_id);

     store(default_store t_states_ids_store) local IdSet local_states_idset;
     local_states_idset = inh_state_id_set;

     store(default_store t_inh_input_ids_store) local IdSet local_inh_input_id_set;
     local_inh_input_id_set = inh_input_id_set;

     store(default_store t_inh_output_ids_store) local IdSet local_inh_output_id_set;
     local_inh_output_id_set = inh_output_id_set;

              store(default_store t_inh_met_store) local time local_inh_met;
     local_inh_met = inh_met;

     store(default_store t_impl_store) local INT is_composite;
     is_composite = Get_Impl_Form(operator_impl);

     store(default_store t_vertex_ids_store) local OpIdSet local_vertex_idset;
     local_vertex_idset = operator_impl.vertex_id_set;

     store(default_store t_edge_ids_store) local IdSet local_edge_idset;
     local_edge_idset = operator_impl.edge_id_set;

     store(default_store t_stream_error_store) local BOOL local_stream_error;
     local_stream_error = (undefined_stream_error || obsolete_stream_error );

     store(default_store t_constraint_error_store) local BOOL local_constraint_error;
     local_constraint_error = (undefined_constraint_error || obsolete_constraint_error
);
   };


/*-------------------------------------------------------------------------------*/
optional_streams:
  Streams{
    type_declarations.inh_types_types = TypeDeclNil;
    type_declarations.inh_generic_types = TypeDeclNil;

    type_declarations.inh_generics_types = TypeDeclNil;
    type_declarations.inh_inputs_types = TypeDeclNil;
    type_declarations.inh_outputs_types = TypeDeclNil;
    type_declarations.inh_states_types = TypeDeclNil;

  };
/*-------------------------------------------------------------------------------*/
bracket_type_declarations
  : BTypeDeclaration
  {
    type_declarations.inh_types_types = TypeDeclNil;
    type_declarations.inh_generic_types = TypeDeclNil;
    type_declarations.inh_generics_types = TypeDeclNil;
    type_declarations.inh_inputs_types = TypeDeclNil;
    type_declarations.inh_outputs_types = TypeDeclNil;
    type_declarations.inh_states_types = TypeDeclNil;
  };
/*-------------------------------------------------------------------------------*/
o_bracket_type_declarations
  : OBTypeDeclaration
  {
    type_declarations.inh_types_types = TypeDeclNil;
    type_declarations.inh_generic_types = TypeDeclNil;
    type_declarations.inh_generics_types = TypeDeclNil;
    type_declarations.inh_inputs_types = TypeDeclNil;
    type_declarations.inh_outputs_types = TypeDeclNil;
    type_declarations.inh_states_types = TypeDeclNil;
  };
/*-------------------------------------------------------------------------------*/

operator_impl
  : OpImplNull, AdaOpImpl
    {$$.vertex_id_set = OpIdSetNil;
     $$.edge_id_set = IdSetNil;
     $$.stream_id_set = IdSetNil;
     $$.constraint_op_id_set = OpIdSetNil;
       $$.syn_vertex_id_met_set = OpIdMetSetNil;
       $$.syn_edge_set          = EdgeSetNil;
       $$.syn_id_constraint_set = IdConstraintSetNil;
    }
  | OperatorImpl
    {$$.vertex_id_set = graph.vertex_id_set;
     $$.edge_id_set = graph.edge_id_set;
     $$.stream_id_set = declarations.stream_id_set;
     $$.constraint_op_id_set = cc.constraint_op_id_set;
       $$.syn_vertex_id_met_set = graph.syn_vertex_id_met_set;
       $$.syn_edge_set          = graph.syn_edge_set;
       $$.syn_id_constraint_set = cc.syn_id_constraint_set;
       cc.inh_vertex_id_met_set = $$.syn_vertex_id_met_set;

/*-------------------*/

       local BOOL producerop_period_le_consumerop_error;
       producerop_period_le_consumerop_error =
         (Is_ProducerOp_Period_LE_ConsumerOp($$.syn_edge_set,
                                             $$.syn_id_constraint_set));

       local STR producerop_period_le_consumerop_msg;
       producerop_period_le_consumerop_msg =
         (producerop_period_le_consumerop_error)
         ?  "\n"
          # "-- !For any edge, the Producer's Period\n"
          # "--  must exceed that of the Consumer."
         :  "";
/*-------------------*/
```

```
      local BOOL sporadic_consumerop_wo_trigger_error;
      sporadic_consumerop_wo_trigger_error =
        (Is_Sporadic_ConsumerOp_WO_Trigger($$.syn_edge_set,
                                           $$.syn_id_constraint_set));

      local STR sporadic_consumerop_wo_trigger_msg;
      sporadic_consumerop_wo_trigger_msg =
        (sporadic_consumerop_wo_trigger_error)
        ?  "\n"
         # "-- !For any edge, if the Consumer is\n"
         # "--  Sporadic, it must have a Trigger."
        :  "";

/*-------------------*/

      local BOOL constr_producerop_and_unconstr_consumerop_w_trigger_error;
      constr_producerop_and_unconstr_consumerop_w_trigger_error =
        (Is_Constr_ProducerOp_And_Unconstr_ConsumerOp_W_Trigger($$.syn_edge_set,

$$.syn_vertex_id_met_set,

$$.syn_id_constraint_set));

      local STR constr_producerop_and_unconstr_consumerop_w_trigger_msg;
      constr_producerop_and_unconstr_consumerop_w_trigger_msg =
        (constr_producerop_and_unconstr_consumerop_w_trigger_error)
        ?  "\n"
         # "-- !For any edge, if the Producer is constrained,\n"
         # "--  an unconstrained Consumer can not have a Trigger."
        :  "";

/*-------------------*/

      local BOOL unconstr_producerop_and_constr_consumerop_w_byall_error;
      unconstr_producerop_and_constr_consumerop_w_byall_error =
        (Is_Unconstr_ProducerOp_And_Constr_ConsumerOp_W_ByAll($$.syn_edge_set,

$$.syn_vertex_id_met_set,

$$.syn_id_constraint_set));

      local STR unconstr_producerop_and_constr_consumerop_w_byall_msg;
      unconstr_producerop_and_constr_consumerop_w_byall_msg =
        (unconstr_producerop_and_constr_consumerop_w_byall_error)
        ?  "\n"
         # "-- !For any edge, if the Producer is unconstrained,\n"
         # "--  a constrained consumer triggered By_All\n"
         # "--  can result in Overflow."
        :  "";

/*-------------------*/

      local BOOL unconstr_producerop_and_constr_consumerop_w_bysome_error;
      unconstr_producerop_and_constr_consumerop_w_bysome_error =
        (Is_Unconstr_ProducerOp_And_Constr_ConsumerOp_W_BySome($$.syn_edge_set,

$$.syn_vertex_id_met_set,

$$.syn_id_constraint_set));

      local STR unconstr_producerop_and_constr_consumerop_w_bysome_msg;
      unconstr_producerop_and_constr_consumerop_w_bysome_msg =
        (unconstr_producerop_and_constr_consumerop_w_bysome_error)
        ?  "\n"
         # "-- !For any edge, if the Producer is unconstrained,\n"
         # "--  a constrained consumer triggered By_Some\n"
         # "--  can result in Data loss."
        :  "";
/*-------------------*/

      local BOOL has_error;
      has_error = (producerop_period_le_consumerop_error ||
                   sporadic_consumerop_wo_trigger_error  ||
                   constr_producerop_and_unconstr_consumerop_w_trigger_error ||
                   unconstr_producerop_and_constr_consumerop_w_byall_error   ||
                   unconstr_producerop_and_constr_consumerop_w_bysome_error);

      local STR error_header;
      error_header = (has_error)
                     ? "\n"
                     # "-------------------------------------------\n"
                     # "-- SCHEDULING NOTICE:"
                     : "";

      local STR error_trailer;
      error_trailer = (has_error)
                      ? "\n"
                      # "-------------------------------------------"
                      : "";


   );

/*-------------------------------------------------------------------------*/
graph
  : GraphNull
    ($$.vertex_id_set = OpIdSetNil;
      $$.syn_vertex_id_met_set = OpIdMetSetNil;
      $$.syn_edge_set          = EdgeSetNil;
     $$.edge_id_set = IdSetNil;
    )
  | Graph
    ($$.vertex_id_set = vertex_list.vertex_id_set;
      $$.syn_vertex_id_met_set = vertex_list.syn_vertex_id_met_set;
      $$.syn_edge_set          = edge_list.syn_edge_set;
     $$.edge_id_set = edge_list.edge_id_set;
    );

/*-------------------------------------------------------------------------*/
vertex_list
  : VertexListNull
    ($$.vertex_id_set = OpIdSetNil;
      $$.syn_vertex_id_met_set = OpIdMetSetNil;
    )
  | VertexListPair
    ($$.vertex_id_set = OpIdSetUnion(a_vertex.vertex_id_set,
vertex_list$2.vertex_id_set);
       $$.syn_vertex_id_met_set =
              Op_Id_Met_Set_Union(Get_Id_Met_Set(a_vertex),
                                  vertex_list$2.syn_vertex_id_met_set);
    );

a_vertex
  : AVertexNull
    ($$.vertex_id_set = OpIdSetNil;
    )
```

```
  | AVertex
    ($$.vertex_id_set = operator_id.vertex_id_set;
    );

operator_id
  : OperatorIdNull
    ($$.vertex_id_set = OpIdSetNil;
    )
  | OperatorId
    ($$.vertex_id_set = SingletonOpIdSet(operator_id);
    );

/*-------------------------------------------------------------------------------*/
edge_list
  : EdgeListNil
    ($$.edge_id_set = IdSetNil;
      $$.syn_edge_set = EdgeSetNil;
    )
  | EdgeListPair
    ($$.edge_id_set = IdSetUnion(an_edge.edge_id_set, edge_list$2.edge_id_set);
      $$.syn_edge_set =
              Edge_Set_Union(Get_Edge_Set(an_edge),
                             edge_list$2.syn_edge_set);
    );

an_edge
  : AnEdgeNull
    ($$.edge_id_set = IdSetNil;
    )
  | AnEdge
    ($$.edge_id_set = SingletonIdSet(id);
    );

/*-------------------------------------------------------------------------------*/
declarations
  : Declarations
    ($$.stream_id_set = optional_streams.stream_id_set;
    );

optional_streams
  : StreamsNull, StreamsPrompt
    ($$.stream_id_set = IdSetNil;
    )
  | Streams
    ($$.stream_id_set = type_declarations.id_set;
    );


/*-------------------------------------------------------------------------------*/
cc
  : CcNull
    ($$.constraint_op_id_set = OpIdSetNil;
      $$.syn_id_constraint_set = IdConstraintSetNil;
    )
  | Cc
    ($$.constraint_op_id_set = constraints.constraint_op_id_set;
      $$.syn_id_constraint_set = constraints.syn_id_constraint_set;
      constraints.inh_vertex_id_met_set = $$.inh_vertex_id_met_set;
    );

constraints
  : ConstraintsNull
    ($$.constraint_op_id_set = OpIdSetNil;
      $$.syn_id_constraint_set = IdConstraintSetNil;
    )
  | ConstraintsPair($$.constraint_op_id_set = OpIdSetUnion(
       a_constraint.constraint_op_id_set,
       constraints$2.constraint_op_id_set);
      $$.syn_id_constraint_set = (Valid_T_Constraint(a_constraint) ||
                                  All_OR_Some_Trigger(a_constraint))
            ? Id_Constraint_Set_Union(Get_Id_Constraint_Set(a_constraint,
$$.inh_vertex_id_met_set),
                                      constraints$2.syn_id_constraint_set)
            : constraints$2.syn_id_constraint_set;

      a_constraint.inh_vertex_id_met_set = $$.inh_vertex_id_met_set;
      constraints$2.inh_vertex_id_met_set = $$.inh_vertex_id_met_set;

    );

a_constraint
  : AConstraintNull
    ($$.constraint_op_id_set = OpIdSetNil;
    )
  | AConstraint
    ($$.constraint_op_id_set = SingletonOpIdSet(operator_id);

      /* An Unconstrained Operator (no MET) should have NO timing constraints. */
      local BOOL unconstrained_op_with_constraints_error;
      unconstrained_op_with_constraints_error =
                        (Partial_Constraint(a_constraint)
                         && !Belong2Constrained_Operator(a_constraint,
                                                         $$.inh_vertex_id_met_set));
      local STR unconstrained_op_with_constraints_msg;
      unconstrained_op_with_constraints_msg =
                (unconstrained_op_with_constraints_error)
                 ? "\n"
                  #"-- !Unconstrained Operators(no MET)\n"
                  #"--  do not have timing constraints."
                 : "";
/*-----------*/

      /* All Constrained Operators must be either Periodic XOR Sporadic. */
      local BOOL invalid_time_constraint_error;
      invalid_time_constraint_error = (!unconstrained_op_with_constraints_error
                                       && Partial_Constraint(a_constraint)
                                       && !Valid_T_Constraint(a_constraint));
      local STR   invalid_time_constraint_msg;
      invalid_time_constraint_msg =
                (invalid_time_constraint_error)
                ? "\n"
                 #"-- !All constrained Operators must be\n"
                 #"--  Periodic(Per & FW) XOR Sporadic(MCP & MRT)"
                 : "";
/*-----------*/

      local BOOL unschedulable_periodic_error;
      unschedulable_periodic_error = (!unconstrained_op_with_constraints_error
                                      && !invalid_time_constraint_error
                                      && Unschedulable_Periodic_Op(a_constraint,
$$.inh_vertex_id_met_set));
      local STR unschedulable_periodic_msg;
      unschedulable_periodic_msg = (unschedulable_periodic_error)
                                    ? "\n"
```

```
                                #"-- !This Periodic Operator is not schedulable."
                                : "";
/*-----------*/

      local BOOL unschedulable_sporadic_error;
      unschedulable_sporadic_error = (!unconstrained_op_with_constraints_error
                                && !invalid_time_constraint_error
                                && Unschedulable_Sporadic_Op(a_constraint,

$$.inh_vertex_id_met_set));
      local STR unschedulable_sporadic_msg;
      unschedulable_sporadic_msg = (unschedulable_sporadic_error)
                                ? "\n"
                                 #"-- !This Sporadic Operator is not schedulable."
                                : "";

/*-----------*/

      local BOOL period_only_error;
      period_only_error = (!unconstrained_op_with_constraints_error
                           && !unschedulable_periodic_error
                           && !unschedulable_sporadic_error
                           && !invalid_time_constraint_error
                           && Period_Only(a_constraint));
      local STR period_only_msg;
      period_only_msg = (period_only_error)
                       ? "\n"
                        #"-- !If left unspecified,\n"
                        #"--  Finish_Within will default to Period."
                       :"";
/*-----------*/

      local BOOL finish_within_only_error;
      finish_within_only_error = (!unconstrained_op_with_constraints_error
                                  && !unschedulable_periodic_error
                                  && !unschedulable_sporadic_error
                                  && !invalid_time_constraint_error
                                  && FinishWithin_Only(a_constraint));
      local STR finish_within_only_msg;
      finish_within_only_msg = (finish_within_only_error)
                              ? "\n"
                               #"-- !If left unspecified,\n"
                               #"--  Period will default to Finish_Within."
                              :"";
/*-----------*/

      local BOOL mrt_only_error;
      mrt_only_error = (!unconstrained_op_with_constraints_error
                        && !unschedulable_periodic_error
                        && !unschedulable_sporadic_error
                        && !invalid_time_constraint_error
                        && MaxResTime_Only(a_constraint));
      local STR mrt_only_msg;
      mrt_only_msg = (mrt_only_error)
                     ? "\n"
                      #"-- !If left unspecified,\n"
                      #"--  MinCallPeriod will default to MaxResTime - MET."
                     :"";
/*-----------*/

      local BOOL mcp_only_error;
      mcp_only_error = (!unconstrained_op_with_constraints_error
                        && !unschedulable_periodic_error
                        && !unschedulable_sporadic_error
                        && !invalid_time_constraint_error
                        && MinCallPeriod_Only(a_constraint));
      local STR mcp_only_msg;
      mcp_only_msg = (mcp_only_error)
                      ? "\n"
                       #"-- !If left unspecified,\n"
                       #"--  MaxResTime will default to MinCallPeriod + MET."
                      :"";

/*-----------*/


      local BOOL has_error;
      has_error = (invalid_time_constraint_error ||
                   unconstrained_op_with_constraints_error ||
                   unschedulable_periodic_error ||
                   unschedulable_sporadic_error ||
                   period_only_error ||
                   finish_within_only_error ||
                   mrt_only_error ||
                   mcp_only_error);

      local STR error_header;
      error_header = (has_error)
                    ? "\n"
                     #"-------------------------------------------\n"
                     #"-- SCHEDULING NOTICE:"
                     : "";

      local STR error_trailer;
      error_trailer = (has_error)
                      ? "\n"
                       #"-------------------------------------------"
                      : "";
   );


/*-------------------------------------------------------------------------*/

component
/* dropped...
  : Op(
   added...*/
   : NoComponent(
       $$.syn_vertex_id_met_set = OpIdMetSetNil;
       $$.syn_id_constraint_set = IdConstraintSetNil;
     )
   | Op(
       $$.syn_vertex_id_met_set = operator_impl.syn_vertex_id_met_set;
       $$.syn_id_constraint_set = operator_impl.syn_id_constraint_set;
     local BOOL is_root;
     is_root = (id != IdNull) &&
           IsElement(id, (prototype.syn_root_ids));


     local STR root_message;
     root_message = (IsElement(id, (prototype.syn_root_ids)))
                                          ? "\n--\n-- This Is A Root Operator"
                                          : "";

     local BOOL multiple_root_error;
     multiple_root_error = (id != IdNull) &&
```

```
                                    IsElement(id,
(prototype.syn_multiple_root_ids));

    local STR multiple_root_message;
    multiple_root_message = (multiple_root_error)
                                    ? "\n--\n-- You have multiple roots\n-- Please
delete the obsolete ones"
                                    : "";

    local BOOL multiple_op_spec_error;
    multiple_op_spec_error = (id != IdNull) &&
          IsElement(id, (prototype.syn_multiple_op_spec));

               local STR multiple_op_spec_message;
               multiple_op_spec_message = (multiple_op_spec_error)
                               ? "\n-- \n-- This Operator Has More Than 1 Operator
Spec\n--     Consistencies will not be enforced \n--    in this operator"
                               : "";

    local BOOL also_defined_as_type_error;
    also_defined_as_type_error = (id != IdNull) &&
          (DefinedType(id, (prototype.syn_defined_types)) > 0);

    local STR also_defined_as_type_message;
    also_defined_as_type_message = (also_defined_as_type_error)
                               ? "\n-- \n-- Id denotes both as Operator and Type"
                               : "";

    local IdSet inh_input_id_set;
    inh_input_id_set = IdSetUnion(
          Extract_Input_Id_Set(id, (prototype.syn_defined_operators)),
          Extract_Input_Id_Set(id, (prototype.syn_defined_types)));

    local IdSet inh_output_id_set;
    inh_output_id_set = IdSetUnion(
          Extract_Output_Id_Set(id, (prototype.syn_defined_operators)),
          Extract_Output_Id_Set(id, (prototype.syn_defined_types)));

    local IdSet input_id_set;
      input_id_set = operator_spec.input_id_set;

    local IdSet output_id_set;
      output_id_set = operator_spec.output_id_set;

    local IdSet undefined_input_id;
    undefined_input_id = IdSetDifference(inh_input_id_set, input_id_set);

    local IdSet obsolete_input_id;
    obsolete_input_id = IdSetDifference(input_id_set, inh_input_id_set);

    local IdSet undefined_output_id;
    undefined_output_id = IdSetDifference(inh_output_id_set, output_id_set);

    local IdSet obsolete_output_id;
    obsolete_output_id = IdSetDifference(output_id_set, inh_output_id_set);

    local BOOL undefined_input_error;
    undefined_input_error = (!IsNull(undefined_input_id));

    local STR undefined_input_message;
    undefined_input_message = (undefined_input_error)
                               ? "\n-- \n-- Missing Input Declarations \n"
                               : "";

               local BOOL obsolete_input_error;
    obsolete_input_error = (!IsNull(obsolete_input_id));

    local STR obsolete_input_message;
    obsolete_input_message = (obsolete_input_error)
                                    ? "\n-- \n-- Obsolete Input Declarations \n"
                                    : "";

    local BOOL undefined_output_error;
    undefined_output_error = (!IsNull(undefined_output_id));

    local STR undefined_output_message;
    undefined_output_message = (undefined_output_error)
                                    ? "\n-- \n-- Missing Output Declarations \n"
                                    : "";

               local BOOL obsolete_output_error;
    obsolete_output_error = (!IsNull(obsolete_output_id));

    local STR obsolete_output_message;
    obsolete_output_message = (obsolete_output_error)
                                    ? "\n-- \n-- Obsolete Output Declarations \n"
                                    : "";

    local type_declarations inh_input_declarations;
    inh_input_declarations = Build_Type_Decl(
            inh_input_id_set,
            (prototype.syn_defined_type_decl));


    local type_declarations inh_output_declarations;
    inh_output_declarations = Build_Type_Decl(
            inh_output_id_set,
            (prototype.syn_defined_type_decl));

    local IdSet input_type_error_set;
    input_type_error_set = Extract_Type_Error_Set(
            IdSetIntersect(input_id_set, inh_input_id_set),
            operator_spec.inputs_types,
            inh_input_declarations);

               local BOOL input_type_error;
    input_type_error = (!IsNull(input_type_error_set));

    local STR input_type_error_message;
    input_type_error_message = (input_type_error)
                                    ? "\n-- \n-- Input Type Declaractions Does Not Match\n-
- Stream Declarations in Parent Operators\n"
                                    : "";

    local IdSet output_type_error_set;
    output_type_error_set = Extract_Type_Error_Set(
            IdSetIntersect(output_id_set, inh_output_id_set),
            operator_spec.outputs_types,
            inh_output_declarations);

               local BOOL output_type_error;
    output_type_error = (!IsNull(output_type_error_set));

    local STR output_type_error_message;
    output_type_error_message = (output_type_error)
                                    ? "\n-- \n-- Output Type Declaractions Does Not
```

```
Match\n-- Stream Declarations in Parent Operators\n"
                                : "";

    local time inh_met;
    inh_met = Extract_Met(
        OperatorId(
          OptionalTypeIdNull,
          id,
          OperatorIdPairsNull),
        (prototype.syn_defined_operators));

    local BOOL met_error;
    met_error = (!multiple_root_error && Different_Time(inh_met,
operator_spec.syn_met));

    local STR met_error_message;
    met_error_message = (met_error)
                                ? "\n-- \n-- Met Does Not Match Vertex Time in Parent
Operator\n"
                                : "";


    local id loc_operator_id;
      loc_operator_id = id;



    local OpIdSet vertex_id_set;
      vertex_id_set = operator_impl.vertex_id_set;

    local IdSet edge_id_set;
      edge_id_set = operator_impl.edge_id_set;

    local IdSet stream_id_set;
      stream_id_set = operator_impl.stream_id_set;

    local OpIdSet constraint_id_set;
      constraint_id_set = operator_impl.constraint_op_id_set;

    local IdSet state_id_set;
      state_id_set = operator_spec.state_id_set;

    local IdSet obsolete_state_id;
    obsolete_state_id = (Get_Impl_Form(operator_impl) == 2)
        ? IdSetDifference(state_id_set, edge_id_set)
        : IdSetNil;

              local BOOL obsolete_state_error;
    obsolete_state_error = (!IsNull(obsolete_state_id));

    local STR obsolete_state_message;
    obsolete_state_message = (obsolete_state_error)
                                ? "\n-- \n-- Obsolete State Declarations \n"
                                : "";


    local IdSet input_output_state_union_set;
      input_output_state_union_set =
        IdSetUnion(state_id_set,
          IdSetUnion(input_id_set, output_id_set));

    local IdSet undefined_stream;
      undefined_stream =
        IdSetDifference(edge_id_set,
          IdSetUnion(stream_id_set,
            input_output_state_union_set));

local IdSet obsolete_stream;
  obsolete_stream =
    IdSetUnion(
      IdSetDifference(stream_id_set, edge_id_set),
      IdSetIntersect(stream_id_set, input_output_state_union_set));

local OpIdSet undefined_constraint;
  undefined_constraint =
      OpIdSetDifference(
        vertex_id_set,
        constraint_id_set);

local OpIdSet obsolete_constraint;
  obsolete_constraint =
      OpIdSetDifference(
        constraint_id_set,
        vertex_id_set);

local BOOL undefined_stream_error;
undefined_stream_error = (!IsNull(undefined_stream));

local STR undefined_stream_message;
undefined_stream_message = (undefined_stream_error)
                              ? "\n-- \n-- Missing Stream Declarations \n"
                              : "";

            local BOOL obsolete_stream_error;
obsolete_stream_error = (!IsNull(obsolete_stream));

local STR obsolete_stream_message;
obsolete_stream_message = (obsolete_stream_error)
                              ? "\n-- \n-- Obsolete Stream Declarations \n"
                              : "";

            local BOOL undefined_constraint_error;
undefined_constraint_error = (!OpIdSetIsNull(undefined_constraint));

local STR undefined_constraint_message;
undefined_constraint_message = (undefined_constraint_error)
                              ? "\n-- \n-- Missing Constraint Entries \n"
                              : "";

            local BOOL obsolete_constraint_error;
obsolete_constraint_error = (!OpIdSetIsNull(obsolete_constraint));

            local STR obsolete_constraint_message;
            obsolete_constraint_message = (obsolete_constraint_error)
                              ? "\n-- \n-- Obsolete Constraint Entries \n"
                              : "";

local BOOL has_error;
has_error = (is_root ||
    multiple_op_spec_error ||
    also_defined_as_type_error ||
            undefined_input_error ||
            obsolete_input_error ||
            undefined_output_error ||
            obsolete_output_error  ||
            input_type_error  ||
```

```
            output_type_error  ||
        undefined_stream_error ||
        obsolete_stream_error ||
        met_error ||
        undefined_constraint_error ||
        obsolete_constraint_error);

            local STR error_header;
            error_header = (has_error)
        ? "\n------------------------------------------ \n-- WARNINGS, ERRORS AND
ALERTS:"
                            : "";


            local STR error_trailer;
            error_trailer = (has_error)
                            ? "\n-- \n------------------------------------------
\n"
                            : "";

    store(default_store operator_id_store) local id operator_name;
    operator_name = id;

    store(default_store multi_op_error_store) local BOOL local_multi_op_error;
    local_multi_op_error = multiple_op_spec_error;

    store(default_store states_ids_store) local IdSet local_states_idset;
    local_states_idset = state_id_set;

    store(default_store inh_input_ids_store) local IdSet local_inh_input_id_set;
    local_inh_input_id_set = inh_input_id_set;

    store(default_store inh_output_ids_store) local IdSet local_inh_output_id_set;
    local_inh_output_id_set = inh_output_id_set;

    store(default_store input_error_store) local BOOL local_input_error;
    local_input_error = (undefined_input_error || obsolete_input_error ||
input_type_error);

    store(default_store output_error_store) local BOOL local_output_error;
    local_output_error = (undefined_output_error || obsolete_output_error ||
output_type_error);

    store(default_store inh_input_decl_store) local type_declarations
local_inh_input_decl;
    local_inh_input_decl = inh_input_declarations;

    store(default_store inh_output_decl_store) local type_declarations
local_inh_output_decl;
    local_inh_output_decl = inh_output_declarations;

    store(default_store met_error_store) local BOOL local_met_error;
    local_met_error = met_error;

    store(default_store inh_met_store) local time local_inh_met;
    local_inh_met = inh_met;

    store(default_store impl_store) local INT is_composite;
    is_composite = Get_Impl_Form(operator_impl);

    store(default_store vertex_ids_store) local OpIdSet local_vertex_idset;
    local_vertex_idset = vertex_id_set;

    store(default_store edge_ids_store) local IdSet local_edge_idset;
    local_edge_idset = edge_id_set;

/*
    store(default_store streams_ids_store) local IdSet local_streams_idset;
    local_streams_idset = IdSetDifference(edge_id_set,
            IdSetUnion(state_id_set,
              IdSetUnion(input_id_set, output_id_set)));


    store(default_store constraints_ids_store) local OpIdSet local_constraints_idset;
    local_constraints_idset = constraint_op_id_set;
*/

    store(default_store stream_error_store) local BOOL local_stream_error;
    local_stream_error = (undefined_stream_error || obsolete_stream_error );

    store(default_store constraint_error_store) local BOOL local_constraint_error;
    local_constraint_error = (undefined_constraint_error || obsolete_constraint_error
);


  )

 | Data(

    $$.syn_vertex_id_met_set = OpIdMetSetNil;
    $$.syn_id_constraint_set = IdConstraintSetNil;
  local id local_type_id;
  local_type_id = id;

  local o_operators syn_o_operators;
  syn_o_operators = type_spec.defined_operators;

  local operator_impl_list syn_operator_impl_list;
  syn_operator_impl_list = type_impl.syn_operator_impl_list;

  local BOOL multiple_type_spec_error;
  multiple_type_spec_error = (id != IdNull) &&
        IsElement(id, (prototype.syn_multiple_type_spec));

          local STR multiple_type_spec_message;
          multiple_type_spec_message = (multiple_type_spec_error)
                          ? "\n-- \n-- Multiply Defined Types "
                          : "";

  local BOOL also_defined_as_op_error;
  also_defined_as_op_error = (id != IdNull) &&
        (DefinedOpr(id, (prototype.syn_defined_operators)) > 0);

  local STR also_defined_as_op_message;
  also_defined_as_op_message = (also_defined_as_op_error)
                          ? "\n-- \n-- Id denotes both as Operator and Type"
                          : "";

  local IdSet undefined_op_impl;
  undefined_op_impl = (PSDLTypeImpl(type_impl)
        ? Extract_Undefined_Op_Impl_In_Data(type_spec.defined_operators,
                type_impl.syn_operator_impl_list)
        : IdSetNil);

  local BOOL undefined_op_impl_error;
  undefined_op_impl_error = (IdSetSize(undefined_op_impl) > 0);
```

```
    local STR undefined_op_impl_message;
    undefined_op_impl_message = (undefined_op_impl_error)
                              ? "\n-- \n-- The Following Operators Has No Operator
Impl:\n"
                              : "";

    local IdSet obsolete_op_impl;
    obsolete_op_impl =
Extract_Obsolete_Op_Impl_In_Data(type_impl.syn_operator_impl_list,
                     type_spec.defined_operators);

    local BOOL obsolete_op_impl_error;
    obsolete_op_impl_error = (IdSetSize(obsolete_op_impl) > 0);

    local STR obsolete_op_impl_message;
    obsolete_op_impl_message = (obsolete_op_impl_error)
                              ? "\n-- \n-- The Following Operators Impl Are
Obsolete:\n"
                              : "";

    local BOOL has_error;
    has_error = (multiple_type_spec_error ||
        also_defined_as_op_error ||
        undefined_op_impl_error ||
        obsolete_op_impl_error
      );

              local STR error_header;
              error_header = (has_error)
        ? "\n------------------------------------------ \n-- WARNINGS, ERRORS AND
ALERTS:"
                              : "";


              local STR error_trailer;
              error_trailer = (has_error)
                              ? "\n-- \n------------------------------------------
\n"
                              : "";

    store(default_store t_undefined_op_impl_store) local IdSet
local_undefined_op_impl_set;
              local_undefined_op_impl_set = undefined_op_impl;

    store(default_store t_obsolete_op_impl_store) local IdSet
local_obsolete_op_impl_set;
              local_obsolete_op_impl_set = obsolete_op_impl;
  );
```

APPENDIX D - Auxiliary Functions

```
/*****************************************************************************
****  This first set of functions were already existing and
****  in support of the basic SDE.
*****************************************************************************/
/*---------------------------------------------------------------------------*/
component GetOperator(component c)
  (with(c)
    (NoComponent:NoComponent,
     Data(i, ts, ti):NoComponent,
     Op(i, os, oi):Op(i, os, oi)
    )
  );

/*---------------------------------------------------------------------------*/
component GetType(component c)
  (with(c)
    (NoComponent:NoComponent,
     Data(i, ts, ti):Data(i, ts, ti),
     Op(i, os, oi):NoComponent
    )
  );

/*---------------------------------------------------------------------------*/

type_declarations Get_Inputs(component c)
  (with(c)
   (  NoComponent:TypeDeclNil,
    Op(i, os, oi):Get_Inputs_Decl_From_Inputs_List(
                                Get_Inputs_List(os)),
    Data(i, ts, ti):
    with (ts)
    (TypeSpec(*, *, oo, *, *, *):
               Get_Inputs_Decl_From_O_Operators(oo)
    )
   )

  );

/*---------------------------------------------------------------------------*/
type_declarations Get_Outputs(component c)
  (with(c)
   (  NoComponent:TypeDeclNil,
    Op(i, os, oi):Get_Outputs_Decl_From_Outputs_List(
                                Get_Outputs_List(os)),
    Data(i, ts, ti):
    with (ts)
    (TypeSpec(*, *, oo, *, *, *):
               Get_Outputs_Decl_From_O_Operators(oo)
    )
   )

  );

/*---------------------------------------------------------------------------*/
type_declarations Get_Inputs_Decl_From_O_Operators(o_operators oo)
  (with(oo)
    (OperatorNil:TypeDeclNil,
     OperatorPair(tos, tail):
     with (tos)
     (TOpSpecNil:Get_Inputs_Decl_From_O_Operators(tail),
      TOpSpec(*, os): Concat_Type_Decl_List(
                  Get_Inputs_Decl_From_Inputs_List(Get_Inputs_List(os)),
                  Get_Inputs_Decl_From_O_Operators(tail))
     )
    )
  );
/*---------------------------------------------------------------------------*/
type_declarations Get_Inputs_Decl_From_Inputs_List(o_inputs_list oil)
  (with(oil)
    (InputsListNone:TypeDeclNil,
     InputsListPair(oi, tail):Concat_Type_Decl_List(
                  Get_Inputs_Decl_From_O_Inputs(oi),
                  Get_Inputs_Decl_From_Inputs_List(tail))
    )
  );
/*---------------------------------------------------------------------------*/
type_declarations Get_Inputs_Decl_From_O_Inputs(o_inputs oi)
  (with(oi)
    (OpInputsNone:TypeDeclNil,
     OpInputs(td, rt):td
    )

  );

/*---------------------------------------------------------------------------*/
type_declarations Get_Outputs_Decl_From_O_Operators(o_operators oo)
  (with(oo)
    (OperatorNil:TypeDeclNil,
     OperatorPair(tos, tail):
     with (tos)
     (TOpSpecNil:Get_Outputs_Decl_From_O_Operators(tail),
      TOpSpec(*, os): Concat_Type_Decl_List(
                  Get_Outputs_Decl_From_Outputs_List(Get_Outputs_List(os)),
                  Get_Outputs_Decl_From_O_Operators(tail))
     )
    )
  );
/*---------------------------------------------------------------------------*/
type_declarations Get_Outputs_Decl_From_Outputs_List(o_outputs_list oil)
  (with(oil)
    (OutputsListNone:TypeDeclNil,
     OutputsListPair(oo, tail):Concat_Type_Decl_List(
                  Get_Outputs_Decl_From_O_Outputs(oo),
                  Get_Outputs_Decl_From_Outputs_List(tail))
    )
  );
/*---------------------------------------------------------------------------*/
type_declarations Get_Outputs_Decl_From_O_Outputs(o_outputs oo)
  (with(oo)
    (OpOutputsNone:TypeDeclNil,
     OpOutputs(td, rt):td
    )

  );
/*---------------------------------------------------------------------------*/
type_declarations Get_Streams(component c)
  (with(c)
   ( NoComponent:TypeDeclNil,
     Op(i, os, oi):Get_Streams_Decl_From_Op_Impl(oi),
     Data(i, ts, ti):
     with (ti)
     (TypeImplNull:TypeDeclNil,
      AdaTypeImpl(*):TypeDeclNil,
      TypeImpl(*, oil):Get_Streams_Decl_From_Op_Impl_List(oil)
     )
   )
```

```
  );
/*-------------------------------------------------------------------------------*/
type_declarations Get_Streams_Decl_From_Op_Impl_List(operator_impl_list oil)
( with (oil)
  ( OpImplListNull:TypeDeclNil,
    OpImplListPair(toi, tail):
    with (toi)
    ( TOpImplNull:Get_Streams_Decl_From_Op_Impl_List(tail),
      TOpImpl(*, oi):Concat_Type_Decl_List(
                    Get_Streams_Decl_From_Op_Impl(oi),
                    Get_Streams_Decl_From_Op_Impl_List(tail))
    )
  )
);

/*-------------------------------------------------------------------------------*/
type_declarations Get_States(component c)
  (with(c)
   ( NoComponent:TypeDeclNil,
     Op(i, os, oi):Get_States_Decl_From_State_List(Get_State_List(os)),
     Data(i, ts, ti):
     with (ts)
     (TypeSpec(*, *, oo, *, *, *):
                Get_States_Decl_From_O_Operators(oo)
     )
   )

  );
/*-------------------------------------------------------------------------------*/
type_declarations Get_States_Decl_From_O_Operators(o_operators oo)
  (with(oo)
    (OperatorNil:TypeDeclNil,
     OperatorPair(tos, tail):
     with (tos)
     (TOpSpecNil:Get_States_Decl_From_O_Operators(tail),
      TOpSpec(*, os): Concat_Type_Decl_List(
                    Get_States_Decl_From_State_List(Get_State_List(os)),
                    Get_States_Decl_From_O_Operators(tail))
     )
    )
  );
/*-------------------------------------------------------------------------------*/
type_declarations Get_States_Decl_From_State_List(o_states_list osl)
  (with(osl)
    (StatesListNone:TypeDeclNil,
     StatesListPair(os, tail):Concat_Type_Decl_List(
                    Get_States_Decl_From_O_State(os),
                    Get_States_Decl_From_State_List(tail))
    )
  );
/*-------------------------------------------------------------------------------*/
type_declarations Get_States_Decl_From_O_State(o_states os)
  (with(os)
    (OpStatesNone:TypeDeclNil,
     OpStates(td, el, rt):td
    )

  );
/*-------------------------------------------------------------------------------*/
vertex_list GetVertices(component c)
  (with(c)
    (NoComponent:VertexListNull,
     Op(i, os, oi):Get_Vertex_List(oi),
     Data(i, ts, ti):
                with (ti)
                (       TypeImplNull:           VertexListNull,
                        AdaTypeImpl(*):         VertexListNull,
                        TypeImpl(*, oil):
Get_Vertex_List_From_Operator_Impl_List(oil)
                )
    )

  );

/*-------------------------------------------------------------------------------*/
vertex_list Concat_Vertex_List(vertex_list vl1, vertex_list vl2)
  (with(vl1)
    (VertexListNull:vl2,
     VertexListPair(av, tail):av::Concat_Vertex_List(tail, vl2)
    )

  );
/*-------------------------------------------------------------------------------*/
edge_list GetEdges(component c)
  (with(c)
    (NoComponent:EdgeListNil,
     Data(i, ts, ti):EdgeListNil,
     Op(i, os, oi):Get_Edge_List(oi)
    )

  );

/*-------------------------------------------------------------------------------*/
edge_list Concat_Edge_List(edge_list el1, edge_list el2)
  (with(el1)
    (EdgeListNil:el2,
     EdgeListPair(ae, tail):ae::Concat_Edge_List(tail, el2)
    )

  );
/*-------------------------------------------------------------------------------*/
INT DefinedOpr(id ident, psdl_components ol)
  (with (ol)
    (PsdlNil:0,
     PsdlPair(c, l): with (c)
      (NoComponent:DefinedOpr(ident, l),
       Data(i, os, oi):DefinedOpr(ident, l),
       Op(i, os, oi):((ident==i)?1:0)+DefinedOpr(ident, l)
      )
    )
  );

/*-------------------------------------------------------------------------------*/
INT DefinedType(id ident, psdl_components ol)
  (with (ol)
    (PsdlNil:0,
     PsdlPair(c, l): with (c)
      (NoComponent:0+DefinedType(ident, l),
       Data(i, os, oi):((i==ident)?1:0)+DefinedType(ident, l),
       Op(i, os, oi):0+DefinedType(ident, l)
      )
    )
  );

/*-------------------------------------------------------------------------------*/
INT DefinedTypeOp(operator_id ident, psdl_components ol)
```

```
( with (ident)
  ( OperatorIdNull:0,
    OperatorId(tid, oid, *):
    with (tid)
    ( OptionalTypeIdNull:0,
      OptionalTypeIdPrompt:0,
      OptionalTypeId(otd):
      with (ol)
      ( PsdlNil: 0,
        PsdlPair(c, l):
        with (c)
        (   NoComponent:DefinedTypeOp(ident, l),
            Data(i, ts, ti):
                 ((i==otd)
                  ? DefinedOpInTypespec(oid, ts)
                  :0)
                 + DefinedTypeOp(ident, l),
            Op(i, os, oi):DefinedTypeOp(ident, l)
        )
      )
    )
  )
);

/*-------------------------------------------------------------------------*/
INT DefinedOpInTypespec(id i, type_spec ts)
( with(ts)
  ( TypeSpec(*, *, oo, *, *, *):OpIsDefinedinTypeSpec(i, oo)
  )
);
/*-------------------------------------------------------------------------*/
INT DefinedVertex(operator_id ident, vertex_list vl)
  (with (vl)
   (VertexListNull:0,
    VertexListPair(av, tail):
    with (av)
     (AVertexNull:DefinedVertex(ident, tail),
      AVertex(oi, *):
      with (oi)
      ( OperatorIdNull:DefinedVertex(ident, tail),
        OperatorId(*, *, *):
            (EqualOpId(oi,ident)?1:0)
                +DefinedVertex(ident, tail)
      )
    )
   )
  );

/*-------------------------------------------------------------------------*/
INT OpIsDefinedinTypeSpec(id i, o_operators oo)
( with (oo)
  ( OperatorNil: 0,
    OperatorPair(tos, tail):
    with(tos)
    ( TOpSpecNil: OpIsDefinedinTypeSpec(i,tail),
      TOpSpec(tid, *): ((i==tid)?1:0)+OpIsDefinedinTypeSpec(i, tail)
    )
  )
);

/*-------------------------------------------------------------------------*/
INT OpIsDefinedinTypeImpl(id i, operator_impl_list oil)
( with (oil)
  ( OpImplListNull: 0,
    OpImplListPair(toi, tail):
    with(toi)
    ( TOpImplNull: OpIsDefinedinTypeImpl(i, tail),
      TOpImpl(t_id, oi):
        ((i==t_id)?1:0)
            + OpIsDefinedinTypeImpl(i, tail)
    )
  )
);

/*-------------------------------------------------------------------------*/
id GetDeclTypeId(decl_type_name t)
{with (t) (
    DTypeSimpleId(i):i,
    DTypeUserDefined(i, *): i,
    DTypeNameNull: IdNull,
    DTypeInteger: IdNull,
    DTypeReal:   IdNull,
    DTypeBoolean: IdNull
/*
    DTypeException: IdNull,
*/
    )
};

/*-------------------------------------------------------------------------*/
/*
 This function counts the number of times that "i" is defined in the id_list
 idl.
*/
/*-------------------------------------------------------------------------*/

INT IdIsDefined(id i, id_list idl)
  (with (idl) (IdNil: 0,
         IdPair(ix, idlx):((i == ix)?1:0)+IdIsDefined(i, idlx)));

/*-------------------------------------------------------------------------*/
INT IdIsDefInTypes(id ident, type_declarations tdl)
{with (tdl)
  (TypeDeclNil: 0,
   TypeDeclPair(adl, tdl2): with(adl)
      (ADeclNil: 0+IdIsDefInTypes(ident, tdl2),
       ADecl(idl, *): IdIsDefined(ident, idl)+IdIsDefInTypes(ident, tdl2)
      )
  )
};

/*-------------------------------------------------------------------------*/
type_declarations GetGenericsTypes(o_generics ogen)
{with(ogen)
  (OpGenericsNone: TypeDeclNil,
   OpGenerics(td, rt): td
  )
};

/*-------------------------------------------------------------------------*/
type_declarations GetInputsTypes(o_inputs oinp)
{with(oinp)
  (OpInputsNone: TypeDeclNil,
   OpInputs(td, rt): td
  )
};
```

APPENDIX D - Auxiliary Functions

```
/*-------------------------------------------------------------------------*/
type_declarations GetOutputsTypes(o_outputs ooutp)
(with(ooutp)
  (OpOutputsNone: TypeDeclNil,
   OpOutputs(td, rt): td
  )
);

/*-------------------------------------------------------------------------*/
type_declarations GetStatesTypes(o_states ostat)
(with(ostat)
  (OpStatesNone: TypeDeclNil,
   OpStates(td, el, rt): td
  )
);

/*-------------------------------------------------------------------------*/
BOOL IsOperator(component c)
{
  with(c)
    (NoComponent:false,
     Data(i, ts, ti):false,
     Op(i, os, oi):true
    )
};

/*-------------------------------------------------------------------------*/
BOOL IsType(component c)
{
        with(c)
                (NoComponent:                   false,
                 Data(i, ts, ti):               true,
                 Op(i, os, oi):                 false
                )
};

/*-------------------------------------------------------------------------*/
id GetId0(component c)
  (with(c) (NoComponent:IdNull,
      Data(i, ts, ti):IdNull,
      Op(i, os, oi): with (i) (IdNull:IdNull,
                                                        Id(v):Id(v))

    )
);
/*-------------------------------------------------------------------------*/
IdSet Extract_Root_Components(psdl_components o, vertex_list vl)
  (with (o)
   (PsdlNil: IdSetNil,
    PsdlPair(c, tail):
    with (c)
    ( NoComponent:Extract_Root_Components(tail, vl),
      Data(*, *, *):Extract_Root_Components(tail, vl),
      Op(i, *, *):
      with (i)
      ( IdNull:  Extract_Root_Components(tail, vl),
        Id(v):   (Operator_Id_Not_In_Vertex_List(
                    OperatorId(
                                OptionalTypeIdNull,
                                i,
                                OperatorIdPairsNull),
                    vl)
                  ? IdSetUnion(
                      SingletonIdSet(i),
                      Extract_Root_Components(tail, vl))
                  : Extract_Root_Components(tail, vl))
      )
    )
   )
  );

/*-------------------------------------------------------------------------*/
IdSet Extract_Undefined_Op(vertex_list vl, psdl_components o)
  (with (vl)
   (VertexListNull:IdSetNil,
              VertexListPair(av, tail):
              with (av)
              (       AVertexNull:    Extract_Undefined_Op(tail, o),
                      AVertex(oi, *):
                      with (oi)
                      (       OperatorIdNull: Extract_Undefined_Op(tail, o),
                              OperatorId(tid, oid, *):
        with (tid)
        (   OptionalTypeIdNull:
                ((DefinedOpr(oid, o) == 0)
                ? oid::Extract_Undefined_Op(tail, o)
                : Extract_Undefined_Op(tail, o)),
            OptionalTypeIdPrompt:
                ((DefinedOpr(oid, o) == 0)
                ? oid::Extract_Undefined_Op(tail, o)
                : Extract_Undefined_Op(tail, o)),
            OptionalTypeId(*):
                Extract_Undefined_Op(tail, o)
        )

      )
    )
   )
  );

/*-------------------------------------------------------------------------*/
OpIdSet Extract_Undefined_Type_Op(vertex_list vl, psdl_components o)
  (with (vl)
   (VertexListNull:OpIdSetNil,
              VertexListPair(av, tail):
              with (av)
              (       AVertexNull:    Extract_Undefined_Type_Op(tail, o),
                      AVertex(oi, *):
                      with (oi)
                      (       OperatorIdNull: Extract_Undefined_Type_Op(tail, o),
                              OperatorId(tid, oid, *):
        with (tid)
        (   OptionalTypeIdNull:
                Extract_Undefined_Type_Op(tail, o),
            OptionalTypeIdPrompt:
                Extract_Undefined_Type_Op(tail, o),
            OptionalTypeId(*):
                ((DefinedTypeOp(oi, o) == 0)
                ? oi::Extract_Undefined_Type_Op(tail, o)
                : Extract_Undefined_Type_Op(tail, o)),
        )

      )
    )
   )
```

```
  };


/*-------------------------------------------------------------------------*/
IdSet Extract_Multiple_Op_Spec_Id(psdl_components o)
  (with (o)
   (PsdlNil: IdSetNil,
    PsdlPair(c, tail):
    with (c)
    ( NoComponent:Extract_Multiple_Op_Spec_Id(tail),
      Data(*, *, *):Extract_Multiple_Op_Spec_Id(tail),
      Op(i, *, *):
      with (i)
      ( IdNull:  Extract_Multiple_Op_Spec_Id(tail),
        Id(v):   ((DefinedOpr(i, tail) > 0)
                 ? IdSetUnion(
                     SingletonIdSet(i),
                     Extract_Multiple_Op_Spec_Id(tail))
                 : Extract_Multiple_Op_Spec_Id(tail))
      )
    )
   )
  );

/*-------------------------------------------------------------------------*/
IdSet Extract_Multiple_Type_Spec_Id(psdl_components o)
  (with (o)
   (PsdlNil: IdSetNil,
    PsdlPair(c, tail):
    with (c)
    ( NoComponent:Extract_Multiple_Type_Spec_Id(tail),
      Data(i, *, *):
      with (i)
      ( IdNull:  Extract_Multiple_Type_Spec_Id(tail),
        Id(v):   ((DefinedType(i, tail) > 0)
                 ? IdSetUnion(
                     SingletonIdSet(i),
                     Extract_Multiple_Type_Spec_Id(tail))
                 : Extract_Multiple_Type_Spec_Id(tail))
      ),

      Op(*, *, *):Extract_Multiple_Type_Spec_Id(tail)
    )
   )
  );

/*-------------------------------------------------------------------------*/
IdSet Extract_Op_Type_Spec_Id(psdl_components o, psdl_components t)
  (with (o)
   (PsdlNil: IdSetNil,
    PsdlPair(c, tail):
    with (c)
    ( NoComponent:Extract_Op_Type_Spec_Id(tail, t),
      Data(*, *, *):Extract_Op_Type_Spec_Id(tail, t),
      Op(i, *, *):
      with (i)
      ( IdNull:  Extract_Op_Type_Spec_Id(tail, t),
        Id(v):   ((DefinedType(i, t) > 0)
                 ? IdSetUnion(
                     SingletonIdSet(i),
                     Extract_Op_Type_Spec_Id(tail, t))
                 : Extract_Op_Type_Spec_Id(tail, t))
      )
    )
   )
  );

/*-------------------------------------------------------------------------*/
OpIdSet Extract_Multiple_Vertices(vertex_list vl)
  (with (vl)
   (VertexListNull:OpIdSetNil,
    VertexListPair(av, tail):
    with (av)
    ( AVertexNull:Extract_Multiple_Vertices(tail),
      AVertex(oi, *):
      with (oi)
      ( OperatorIdNull: Extract_Multiple_Vertices(tail),
        OperatorId(*, *, *):
             ((DefinedVertex(oi, tail) > 0)
             ? OpIdSetUnion(
                 SingletonOpIdSet(oi),
                 Extract_Multiple_Vertices(tail))
             : Extract_Multiple_Vertices(tail))
      )
    )
   )
  );

/*-------------------------------------------------------------------------*/
IdSet Extract_Multiple_TD(type_declarations td)
  (with (td)
   (TypeDeclNil:IdSetNil,
    TypeDeclPair(ad, tail):
    with (ad)
    ( ADeclNil:Extract_Multiple_TD(tail),
      ADecl(idl, *):
        IdSetUnion(
            Extract_Multiple_Ids_In_Idl_N_TD(idl, tail),
            Extract_Multiple_TD(tail))
    )
   )
  );

/*-------------------------------------------------------------------------*/
IdSet Extract_Multiple_Ids_In_Idl_N_TD(id_list idl, type_declarations td)
  (with (idl)
   (IdNil:   IdSetNil,
    IdPair(i, idl_tail):
    with (i)
    ( IdNull:Extract_Multiple_Ids_In_Idl_N_TD(idl_tail, td),
      Id(*):
        ((IdIsDefined(i, idl_tail) + IdIsDefInTypes(i, td)) > 0)
        ? IdSetUnion(
            SingletonIdSet(i),
            Extract_Multiple_Ids_In_Idl_N_TD(idl_tail, td))
        : Extract_Multiple_Ids_In_Idl_N_TD(idl_tail, td)
    )
   )
  );


/*-------------------------------------------------------------------------*/
IdSet Extract_Type_Error_Set(IdSet ids, type_declarations td1, type_declarations td2)
  (with(ids)
   (IdSetNil:IdSetNil,
```

```
    IdSetPair(i, tail):(Identical_Type_Name(
                Find_Type_Name(i, td1),
                Find_Type_Name(i, td2))
            ? Extract_Type_Error_Set(tail, td1, td2)
            : i::Extract_Type_Error_Set(tail, td1, td2))
  )
 );

/*-------------------------------------------------------------------------*/
IdSet Extract_Undefined_Op_Impl_In_Data(o_operators oo, operator_impl_list oil)
 (with (oo)
  (OperatorNil:IdSetNil,
            OperatorPair(tos, tail):
            with (tos)
            (       TOpSpecNil:    Extract_Undefined_Op_Impl_In_Data(tail, oil),
                    TOpSpec(tid, *):
                    with (tid)
                    (       IdNull:Extract_Undefined_Op_Impl_In_Data(tail, oil),
                            Id(*):
             ((OpIsDefinedinTypeImpl(tid, oil) == 0)
             ? tid::Extract_Undefined_Op_Impl_In_Data(tail, oil)
             : Extract_Undefined_Op_Impl_In_Data(tail, oil))
     )
   )
  )
 );

/*-------------------------------------------------------------------------*/
IdSet Extract_Obsolete_Op_Impl_In_Data(operator_impl_list oil, o_operators oo)
 (with (oil)
  (OpImplListNull:IdSetNil,
            OpImplListPair(toi, tail):
            with (toi)
            (       TOpImplNull:    Extract_Obsolete_Op_Impl_In_Data(tail, oo),
                    TOpImpl(oid, *):
                    with (oid)
                    (       IdNull:Extract_Obsolete_Op_Impl_In_Data(tail, oo),
                            Id(*):
             ((OpIsDefinedinTypeSpec(oid, oo) == 0)
             ? oid::Extract_Obsolete_Op_Impl_In_Data(tail, oo)
             : Extract_Obsolete_Op_Impl_In_Data(tail, oo))
     )
   )
  )
 );

/*-------------------------------------------------------------------------*/
IdSet Get_Input_Id_Set_From_TOpSpec(t_oper_spec tos)
{ with (tos)
  ( TOpSpecNil:IdSetNil,
    TOpSpec(*, os):
    with(os)
    ( OperatorSpec(g, i, o, s, e, t, k, inf, for):
                Make_Input_Id_Set(i)
    )
  )
);
/*-------------------------------------------------------------------------*/
IdSet Get_Output_Id_Set_From_TOpSpec(t_oper_spec tos)
{ with (tos)
  ( TOpSpecNil:IdSetNil,
    TOpSpec(*, os):
    with(os)
    ( OperatorSpec(g, i, o, s, e, t, k, inf, for):
                Make_Output_Id_Set(o)
    )
  )
);
/*-------------------------------------------------------------------------*/
IdSet Get_State_Id_Set_From_TOpSpec(t_oper_spec tos)
{ with (tos)
  ( TOpSpecNil:IdSetNil,
    TOpSpec(*, os):
    with(os)
    ( OperatorSpec(g, i, o, s, e, t, k, inf, for):
                Make_State_Id_Set(s)
    )
  )
);
/*-------------------------------------------------------------------------*/
time Get_Met_From_TOpSpec(t_oper_spec tos)
{ with (tos)
  ( TOpSpecNil:TimeNull,
    TOpSpec(*, os):
    with(os)
    ( OperatorSpec(g, i, o, s, e, t, k, inf, for):
      with (t)
      ( OpTimingInfoNone:TimeNull,
        OpTimingInfoPrompt:TimeNull,
        OpTimingInfo(ti, *): ti
      )
    )
  )
);
/*-------------------------------------------------------------------------*/
IdSet Make_Input_Id_Set(o_inputs_list ol)
{
  with (ol)
  ( InputsListNone:IdSetNil,
    InputsListPair(oi, tail):
    with (oi)
    ( OpInputsNone:Make_Input_Id_Set(tail),
      OpInputs(td, rt):
        IdSetUnion(
            Make_Id_Set_From_Type_Decl(td),
            Make_Input_Id_Set(tail))
    )
  )
);
/*-------------------------------------------------------------------------*/
IdSet Make_Output_Id_Set(o_outputs_list ol)
{
  with (ol)
  ( OutputsListNone:IdSetNil,
    OutputsListPair(oo, tail):
    with (oo)
    ( OpOutputsNone:Make_Output_Id_Set(tail),
      OpOutputs(td, rt):
        IdSetUnion(
            Make_Id_Set_From_Type_Decl(td),
            Make_Output_Id_Set(tail))
    )
  )
);
/*-------------------------------------------------------------------------*/
IdSet Make_State_Id_Set(o_states_list ol)
```

```
{
  with (ol)
  ( StatesListNone:IdSetNil,
    StatesListPair(os, tail):
    with (os)
    ( OpStatesNone:Make_State_Id_Set(tail),
      OpStates(td, el, rt):
        IdSetUnion(
            Make_Id_Set_From_Type_Decl(td),
            Make_State_Id_Set(tail))
    )
  )
};
/*-------------------------------------------------------------------------*/
IdSet Make_Id_Set_From_Type_Decl(type_declarations td)
{
  with (td)
  ( TypeDeclNil:IdSetNil,
    TypeDeclPair(ad, tail):
    with (ad)
    ( ADeclNil:Make_Id_Set_From_Type_Decl(tail),
      ADecl(idl, dtn):
        IdSetUnion(
            Make_Id_Set_From_Id_List(idl),
            Make_Id_Set_From_Type_Decl(tail))
    )
  )
};
/*-------------------------------------------------------------------------*/
IdSet Make_Id_Set_From_Id_List(id_list idl)
{
  with (idl)
  ( IdNil:IdSetNil,
    IdPair(id, tail):
      IdSetUnion(
        SingletonIdSet(id),
        Make_Id_Set_From_Id_List(tail))
  )
};
/*-------------------------------------------------------------------------*/
BOOL Identical_Type_Name(decl_type_name n1, decl_type_name n2)
  (with(n1)
   (DTypeNameNull:
    with(n2)
    ( DTypeNameNull:true,
      DTypeInteger:false,
      DTypeReal:false,
      DTypeBoolean:false,
/*
      DTypeException:false,
*/
      DTypeSimpleId(*):false,
      DTypeUserDefined(*,*):false
    ),

    DTypeInteger:
    with(n2)
    ( DTypeNameNull:false,
      DTypeInteger:true,
      DTypeReal:false,
      DTypeBoolean:false,
/*
      DTypeException:false,
*/
      DTypeSimpleId(*):false,
      DTypeUserDefined(*,*):false
    ),

    DTypeReal:
    with(n2)
    ( DTypeNameNull:false,
      DTypeInteger:false,
      DTypeReal:true,
      DTypeBoolean:false,
/*
      DTypeException:false,
*/
      DTypeSimpleId(*):false,
      DTypeUserDefined(*,*):false
    ),

    DTypeBoolean:
    with(n2)
    ( DTypeNameNull:false,
      DTypeInteger:false,
      DTypeReal:false,
      DTypeBoolean:true,
/*
      DTypeException:false,
*/
      DTypeSimpleId(*):false,
      DTypeUserDefined(*,*):false
    ),

/*
    DTypeException:
    with(n2)
    ( DTypeNameNull:false,
      DTypeInteger:false,
      DTypeReal:false,
      DTypeBoolean:false,
      DTypeException:true,
      DTypeSimpleId(*):false,
      DTypeUserDefined(*,*):false
    ),
*/


    DTypeSimpleId(i1):
    with(n2)
    ( DTypeNameNull:false,
      DTypeInteger:false,
      DTypeReal:false,
      DTypeBoolean:false,
/*
      DTypeException:false,
*/
      DTypeSimpleId(i2):(i1 == i2),
      DTypeUserDefined(*,*):false
    ),

    DTypeUserDefined(i1, *):
    with(n2)
    ( DTypeNameNull:false,
      DTypeInteger:false,
      DTypeReal:false,
```

```
      DTypeBoolean:false,
/*
      DTypeException:false,
*/
      DTypeSimpleId(*):false,
      DTypeUserDefined(i2,*):(i1 == i2)
    )
   )
  );

/*-------------------------------------------------------------------------*/

/*************************************************************************
****  This second set of functions were already existing and
****  in for creating the interaction between the SYNGEN and the
****  Graphic Editor and Graphic Viewer.
****  Documented originally as file: ed.01.ssl
*************************************************************************/
/*-------------------------------------------------------------------------*/
list IdSet;
IdSet
  : exported IdSetNil()
  | exported IdSetPair(id IdSet)
  ;

IdSet exported NullSet() { IdSetNil };

BOOL exported IsNull(IdSet s) { s == IdSetNil };

BOOL exported IsElement(id i, IdSet s) {
  with (s)
  ( IdSetNil: false,
    IdSetPair(head, t):((i != head)
             ? IsElement(i, t)
             : true)
/*
    IdSetPair(head, t):((i < head)
             ? IsElement(i, t)
             : ((i == head)
               ? true
               : false))
*/
  )
};

IdSet exported SingletonIdSet(id i) {
  with (i)
  (
    IdNull :IdSetNil,
    Id(*) :i :: IdSetNil
  )
};

IdSet exported IdSetUnion(IdSet s1, IdSet s2) {
  with (s1)
  ( IdSetNil: s2,
    IdSetPair(i1, t1):
    with (s2)
    ( IdSetNil: s1,
      IdSetPair(i2, t2)
        : i1 < i2 ?i1 :: IdSetUnion(t1, s2)
        : i1 == i2 ?i1 :: IdSetUnion(t1, t2)
        :       i2 :: IdSetUnion(s1, t2)
    )
  )
};

IdSet exported IdSetIntersect(IdSet s1, IdSet s2) {
  with (s1)
  ( IdSetNil: IdSetNil,
    IdSetPair(i1, t1):
    with (s2)
    ( IdSetNil: IdSetNil,
      IdSetPair(i2, t2)
        : i1 < i2 ?IdSetIntersect(t1, s2)
        : i1 == i2 ?i1 :: IdSetIntersect(t1, t2)
        :       IdSetIntersect(s1, t2)
    )
  )
};

IdSet exported IdSetDifference(IdSet s1, IdSet s2) {
  with (s1)
  ( IdSetNil: s1,
    IdSetPair(i1, t1):
    with (s2)
    ( IdSetNil: s1,
      IdSetPair(i2, t2)
        : i1 < i2 ?i1 :: IdSetDifference(t1, s2)
        : i1 == i2 ?IdSetDifference(t1, t2)
        :       IdSetDifference(s1, t2)
    )
  )
};

id exported FirstElement(IdSet s) {
  with (s)
  ( IdSetNil: IdNull,
    IdSetPair(i1, t1): i1
  )
};

IdSet exported IdSetTail(IdSet s) {
  with (s)
  ( IdSetNil: IdSetNil,
    IdSetPair(i1, t1): t1
  )
};

INT exported IdSetSize(IdSet s) {
  with (s)
  ( IdSetNil: 0,
    IdSetPair(i1, t1): 1 + IdSetSize(t1)
  )
};

/*-------------------------------------------------------------------------*/
list OpIdSet;
OpIdSet
  : exported OpIdSetNil()
  | exported OpIdSetPair(operator_id OpIdSet)
  ;

OpIdSet exported NullOpIdSet() { OpIdSetNil };

BOOL exported OpIdSetIsNull(OpIdSet s) { s == OpIdSetNil };
```

APPENDIX D - Auxiliary Functions

```
BOOL exported OpIdIsInOpIdSet(operator_id i, OpIdSet s) {
  with (s)
  { OpIdSetNil: false,
    OpIdSetPair(oid, t):(LessThanOpId(i, oid)
             ? OpIdIsInOpIdSet(i, t)
             : (EqualOpId(i, oid)
               ? true
               : false))
  }
};

OpIdSet exported SingletonOpIdSet(operator_id i) {
  with (i)
  {
    OperatorIdNull :OpIdSetNil,
    OperatorId(*, *, *) :i :: OpIdSetNil
  }
};

OpIdSet exported OpIdSetUnion(OpIdSet s1, OpIdSet s2) {
  with (s1)
  { OpIdSetNil: s2,
    OpIdSetPair(i1, t1):
    with(s2)
    { OpIdSetNil:s1,
      OpIdSetPair(i2, t2):
             (LessThanOpId(i1, i2)
             ? i1::OpIdSetUnion(t1, s2)
             : (EqualOpId(i1, i2)
               ? i1::OpIdSetUnion(t1, t2)
               : i2::OpIdSetUnion(s1, t2)))
    }
  }
};


OpIdSet exported OpIdSetIntersect(OpIdSet s1, OpIdSet s2) {
  with (s1)
  { OpIdSetNil: OpIdSetNil,
    OpIdSetPair(i1, t1):
    with (s2)
    { OpIdSetNil: OpIdSetNil,
      OpIdSetPair(i2, t2):
             (LessThanOpId(i1, i2)
             ? OpIdSetIntersect(t1, s2)
             : (EqualOpId(i1, i2)
               ? i1::OpIdSetIntersect(t1, t2)
               : OpIdSetIntersect(s1, t2)))
    }
  }
};


OpIdSet exported OpIdSetDifference(OpIdSet s1, OpIdSet s2) {
  with (s1)
  { OpIdSetNil:OpIdSetNil,
    OpIdSetPair(i1, t1):
    with (s2)
    { OpIdSetNil: s1,
      OpIdSetPair(i2, t2):
             (LessThanOpId(i1, i2)
             ? i1::OpIdSetDifference(t1, s2)
             : (EqualOpId(i1, i2)
               ? OpIdSetDifference(t1, t2)
               : OpIdSetDifference(s1, t2)))
    }
  }
};


operator_id exported OpIdSetFirstElement(OpIdSet s) {
  with (s)
  { OpIdSetNil:OperatorIdNull,
    OpIdSetPair(i1, t1):i1
  }
};

OpIdSet exported OpIdSetTail(OpIdSet s) {
  with (s)
  { OpIdSetNil: OpIdSetNil,
    OpIdSetPair(i1, t1): t1
  }
};

INT exported OpIdSetSize(OpIdSet s) {
  with (s)
  { OpIdSetNil: 0,
    OpIdSetPair(i1, t1): 1 + OpIdSetSize(t1)
  }
};

BOOL exported EqualOpId(operator_id i1, operator_id i2)
{ with (i1)
  { OperatorIdNull:
    with (i2)
    { OperatorIdNull:true,
      OperatorId(*, *, *): false
    },

    OperatorId(tid_1, id_1, oip_1):
    with (i2)
    { OperatorIdNull:false,
      OperatorId(tid_2, id_2, oip_2):
             (EqualTypeId(tid_1, tid_2) &&
                 (id_1 == id_2) &&
                 EqualOperatorIdPairs(
                     oip_1, oip_2))
    }
  }
};

BOOL EqualTypeId(optional_type_id tid_1, optional_type_id tid_2)
{ with (tid_1)
  { OptionalTypeIdNull:
    with (tid_2)
    { OptionalTypeIdNull:true,
      OptionalTypeIdPrompt:true,
      OptionalTypeId(*):false
    },

    OptionalTypeIdPrompt:
    with (tid_2)
    { OptionalTypeIdNull:true,
      OptionalTypeIdPrompt:true,
      OptionalTypeId(*):false
```

```
    ),

    OptionalTypeId(id_1):
    with (tid_2)
    ( OptionalTypeIdNull:true,
      OptionalTypeIdPrompt:true,
      OptionalTypeId(id_2):id_1 == id_2
    )
  )
};

BOOL EqualOperatorIdPairs(operator_id_pairs oip_1, operator_id_pairs oip_2)
{ with (oip_1)
  ( OperatorIdPairsNull:
    with (oip_2)
    ( OperatorIdPairsNull:true,
      OperatorIdPairsPrompt:true,
      OperatorIdPairs(*,*):false
    ),

    OperatorIdPairsPrompt:
    with (oip_2)
    ( OperatorIdPairsNull:true,
      OperatorIdPairsPrompt:true,
      OperatorIdPairs(*,*):false
    ),

    OperatorIdPairs(ail_11, ail_12):
    with (oip_2)
    ( OperatorIdPairsNull:false,
      OperatorIdPairsPrompt:false,
      OperatorIdPairs(ail_21, ail_22):
            (EqualAIDLIST(ail_11, ail_21)
             && EqualAIDLIST(ail_12, ail_22))
    )
  )
};

BOOL EqualAIDLIST(alone_id_list ail_1, alone_id_list ail_2)
{ with(ail_1)
  ( AIdNil:
    with (ail_2)
    ( AIdNil:    true,
      AIdPair(*,*):false
    ),

    AIdPair(id_1, tail_1):
    with (ail_2)
    ( AIdNil:    false,
      AIdPair(id_2, tail_2):
            ((id_1 == id_2) &&
             EqualAIDLIST(tail_1, tail_2))
    )
  )
};


BOOL LessThanOpId(operator_id i1, operator_id i2)
{ with (i1)
  ( OperatorIdNull:
    with (i2)
    ( OperatorIdNull:false,
      OperatorId(*, *, *): true
    ),

    OperatorId(tid_1, id_1, oip_1):
    with (i2)
    ( OperatorIdNull:false,
      OperatorId(tid_2, id_2, oip_2):
        (LessThanTypeId(tid_1, tid_2) ||
            (EqualTypeId(tid_1, tid_2) &&
                ((id_1 < id_2) ||
                    ((id_1 == id_2) &&
                 LessThanOperatorIdPairs(oip_1, oip_2)
                    )
                )
            )
        )
    )
  )
};

BOOL LessThanTypeId(optional_type_id tid_1, optional_type_id tid_2)
{ with (tid_1)
  ( OptionalTypeIdNull:
    with (tid_2)
    ( OptionalTypeIdNull:false,
      OptionalTypeIdPrompt:false,
      OptionalTypeId(*):true
    ),

    OptionalTypeIdPrompt:
    with (tid_2)
    ( OptionalTypeIdNull:false,
      OptionalTypeIdPrompt:false,
      OptionalTypeId(*):true
    ),

    OptionalTypeId(id_1):
    with (tid_2)
    ( OptionalTypeIdNull:false,
      OptionalTypeIdPrompt:false,
      OptionalTypeId(id_2):id_1 < id_2
    )
  )
};

BOOL LessThanOperatorIdPairs(operator_id_pairs oip_1, operator_id_pairs oip_2)
{ with (oip_1)
  ( OperatorIdPairsNull:
    with (oip_2)
    ( OperatorIdPairsNull:false,
      OperatorIdPairsPrompt:false,
      OperatorIdPairs(*,*):true
    ),

    OperatorIdPairsPrompt:
    with (oip_2)
    ( OperatorIdPairsNull:false,
      OperatorIdPairsPrompt:false,
      OperatorIdPairs(*,*):true
    ),

    OperatorIdPairs(ail_11, ail_12):
    with (oip_2)
    ( OperatorIdPairsNull:false,
```

```
      OperatorIdPairsPrompt:false,
      OperatorIdPairs(ail_21, ail_22):
            (LessThanAIDLIST(ail_11, ail_21) ||
                (EqualAIDLIST(ail_11, ail_21) &&
                   LessThanAIDLIST(ail_12, ail_22)
                )
            )
    )
  )
};

BOOL LessThanAIDLIST(alone_id_list ail_1, alone_id_list ail_2)
{ with(ail_1)
  ( AIdNil:
    with (ail_2)
    ( AIdNil:    false,
      AIdPair(*,*):true
    ),

    AIdPair(id_1, tail_1):
    with (ail_2)
    ( AIdNil:    false,
      AIdPair(id_2, tail_2):
            ((id_1 < id_2) ||
                ((id_1 == id_2) &&
                    LessThanAIDLIST(tail_1, tail_2)
                )
            )
    )
  )
};

/*-------------------------------------------------------------------------------*/
list state_list;
state_list
        : exported State_ListNil()
        | exported State_listPair(a_state_pair state_list)
        ;

/* (id id) = (operator_id type_id) */
a_state_pair
        : exported A_State_Pair(id id);

list coor_list;
coor_list
        : exported Coor_ListNil()
        | exported Coor_ListPair(a_coordinate coor_list)
        ;

a_coordinate
        : exported XYValues(INT INT);
/*-------------------------------------------------------------------------------*/
list id_to_met_list;
id_to_met_list
        : exported IdToMetListNil()
        | exported IdToMetListPair(a_pair id_to_met_list);

a_pair
        : NumberNamePair(id INT);
/*-------------------------------------------------------------------------------*/
list vtx_met_list;
vtx_met_list
        : VtxMetListNil()
        | VtxMetListPair(vtx_met_pair vtx_met_list);

vtx_met_pair
        : VtxMetPair(id INT);
/*-------------------------------------------------------------------------------*/
INT exported IdIsNull(id i)
{
  with(i) (IdNull: 1,
                  Id(*):  0)
};
/*-------------------------------------------------------------------------------*/
INT exported OperatorIdIsNull(operator_id i)
{
  with(i) (OperatorIdNull: 1,
           OperatorId(*, *, *):  0)
};
/*-------------------------------------------------------------------------------*/
INT exported VtxListIsNil(vtx_met_list l)
{
  with(l) (VtxMetListNil: 1,
                  VtxMetListPair(*, *): 0)
};
/*-------------------------------------------------------------------------------*/
INT exported AVertexIsNull(a_vertex v)
{
  with(v) (AVertexNull: 1,
                  AVertex(*, *): 0)
};
/*-------------------------------------------------------------------------------*/
STR exported Get_Op_Name(component p)
{
  with(p)  (
            Op(i, *, *): Get_Id(i),
            NoComponent: "",
            Data(*, *, *): ""
            )
};
/*-------------------------------------------------------------------------------*/
operator_spec exported Get_Operator_Spec(component p)
{
  with(p)  (
            Op(*, os, *): os
           )
};
/*-------------------------------------------------------------------------------*/
operator_impl exported Get_Operator_Impl(component p)
{
  with(p)  (
            Op(*, *, oi): oi,
            NoComponent: OpImplNull,
            Data(*, *, *): OpImplNull
            )
};
/*-------------------------------------------------------------------------------*/
INT exported Get_Impl_Form(operator_impl oi)
{
  with(oi)
               (OpImplNull:    0,
                AdaOpImpl(*):   0,
                OperatorImpl(g, *, *):
                       with(g)
                               (GraphNull: 1,
                                Graph(*, *) : 2)
```

```
                )
};
/*-------------------------------------------------------------------------*/
BOOL exported Op_Impl_Is_Null(operator_impl oi)
{
  with (oi)
                (OpImplNull:    true,
                 AdaOpImpl(*):   false,
                 OperatorImpl(*, *, *): false
    )
};
/*-------------------------------------------------------------------------*/
BOOL exported IsTypeDeclNil(type_declarations td)
{
  with (td)
    (TypeDeclNil: true,
     TypeDeclPair(*,*): false
    )
};

/*-------------------------------------------------------------------------*/
BOOL exported IsADeclNil(a_decl ad)
{
  with (ad)
    (ADeclNil: true,
     ADecl(*,*): false
    )
};

/*-------------------------------------------------------------------------*/
BOOL exported IsIdListNil(id_list idl)
{
  with (idl)
    (IdNil: true,
     IdPair(*,*): false
    )
};

/*-------------------------------------------------------------------------*/
id exported Get_Id_From_Id_List(id_list idl)
{
  with (idl)
    (IdNil: IdNull,
     IdPair(id,*): id
    )
};

/*-------------------------------------------------------------------------*/
constraints exported Get_Constraints_From_Op_Impl(operator_impl oi)
{
  with(oi)
                (OpImplNull:    ConstraintsNull,
                 AdaOpImpl(*):   ConstraintsNull,
                 OperatorImpl(*, *, c):
                        with(c)
                                (CcNull: ConstraintsNull,
                                 Cc(cts, *) : cts)
                )
};

/*-------------------------------------------------------------------------*/
optional_streams exported Make_StreamsNull()
{
  StreamsNull
);

/*-------------------------------------------------------------------------*/
constraints exported Make_ConstraintsNull()
{
  ConstraintsNull
);

/*-------------------------------------------------------------------------*/
/* the following routine is commented out because it is not longer being used
a_constraint exported Make_AConstraint_From_Operator_Id(operator_id i)
{
  AConstraint(i,
      OptionalTriggerNull,
                        OptPeriodNull,
                        OptFinishWithinNull,
                        OptMcpNull,
                        OptMrtNull,
                        OutputGuardsNil,
                        ExceptionOpsNull,
                        TimerOperationsNil)

);
*/

/*-------------------------------------------------------------------------*/
a_constraint exported Make_AConstraintNull()
{
  AConstraintNull
);

/*-------------------------------------------------------------------------*/
id exported Make_Id_From_SSLstring(STR x)
{
  Id(x)
);

/*-------------------------------------------------------------------------*/
id exported Make_IdNull()
{
  IdNull
);

/*-------------------------------------------------------------------------*/
id_list exported Make_Id_List_Nil()
{
  IdNil
);

/*-------------------------------------------------------------------------*/
id_list exported Make_Id_List(id i, id_list idl)
{
  IdPair(i, idl)
);

/*-------------------------------------------------------------------------*/
a_decl exported Make_A_Decl_Nil()
{
  ADeclNil
);

/*-------------------------------------------------------------------------*/
```

APPENDIX D - Auxiliary Functions

```
a_decl exported Make_A_Decl_From_Id(id i)
{
  ADecl(IdPair(i, IdNil), DTypeNameNull)
};

/*-------------------------------------------------------------------------------*/
a_decl exported Make_A_Decl_Pair(id_list idl, decl_type_name dtn)
{
  ADecl(idl, dtn)
};

/*-------------------------------------------------------------------------------*/
type_declarations exported Get_Streams_Decl_From_Op_Impl(operator_impl oi)
{
  with(oi)
                (OpImplNull:    TypeDeclNil,
                 AdaOpImpl(*):   TypeDeclNil,
                 OperatorImpl(*, d, *):
                        with(d)
                                ( /* DeclarationsNull: TypeDeclNil, */
                                 Declarations(os, *) :
         with(os)
             (StreamsNull: TypeDeclNil,
              StreamsPrompt: TypeDeclNil,
              Streams(td): td
             )
        )
                )
};

/*-------------------------------------------------------------------------------*/
BOOL exported Op_Impl_Has_Non_Null_Streams(operator_impl oi)
{
  with(oi)
                (OpImplNull:    false,
                 AdaOpImpl(*):   false,
                 OperatorImpl(*, d, *):
                        with(d)
                                ( /* DeclarationsNull: false, */
                                 Declarations(os, *) :
         with(os)
             (StreamsNull: false,
              StreamsPrompt: false,
              Streams(*): true
             )
        )
                )
};

/*-------------------------------------------------------------------------------*/
BOOL exported Op_Impl_Has_Non_Null_Declarations(operator_impl oi)
{
  with(oi)
                (OpImplNull:    false,
                 AdaOpImpl(*):   false,
                 OperatorImpl(*, d, *):
                        with(d)
                                ( /* DeclarationsNull: false, */
                                 Declarations(*, *) : true)
                )
};

/*-------------------------------------------------------------------------------*/
```

```
BOOL exported Op_Impl_Has_Non_Null_Cc(operator_impl oi)
{
  with(oi)
                (OpImplNull:    false,
                 AdaOpImpl(*):   false,
                 OperatorImpl(*, *, c):
                        with(c)
                                (CcNull: false,
                                 Cc(*, *) : true)
                )
};

/*-------------------------------------------------------------------------------*/
BOOL exported Op_Impl_Has_Non_Null_Constraints(operator_impl oi)
{
  with(oi)
                (OpImplNull:    false,
                 AdaOpImpl(*):   false,
                 OperatorImpl(*, *, c):
                 with(c)
       (CcNull: false,
        Cc(c, *) :
        with(c)
          (ConstraintsNull: false,
           ConstraintsPair(*,*): true
          )
       )
                )
};

/*-------------------------------------------------------------------------------*/
BOOL exported IsAConstraintNull(a_constraint ac)
{ with (ac)
  (AConstraintNull: true,
   AConstraint(*, *, *, *, *, *, *, *, *): false
  )
};

/*-------------------------------------------------------------------------------*/
BOOL exported IsConstraintsNull(constraints c)
{
  with (c)
    (ConstraintsNull: true,
     ConstraintsPair(*,*): false
    )
};

/*-------------------------------------------------------------------------------*/
BOOL exported Valid_Met(time t)
{
  with(t)
  ( TimeNull: false,
    Time(*, *): true
      )

};
/*-------------------------------------------------------------------------------*/
time exported Get_Met_From_Op_Spec(operator_spec p)
{
  with(p)
        (       OperatorSpec(*, *, *, *, *, t, *, *, *):
                with (t)
    ( OpTimingInfoNone: TimeNull,
```

```
      OpTimingInfoPrompt: TimeNull,
      OpTimingInfo(t, *): t
    )
  )
};
/*-------------------------------------------------------------------------------*/
time exported Extract_Met(operator_id i, psdl_components o)
{
  with(o)
  ( PsdlNil:TimeNull,
    PsdlPair(c, tail):((Operator_Id_Not_In_Vertex_List(i, GetVertices(c))))
            ? Extract_Met(i, tail)
            : Get_Vertex_Time(Get_Vertex_With_Operator_Id(i, GetVertices(c)))
  )
};

/*-------------------------------------------------------------------------------*/
BOOL exported Different_Time(time t1, time t2)
{
  (Convert_Time_To_Integer(t1) != Convert_Time_To_Integer(t2))
};

/*-------------------------------------------------------------------------------*/
a_vertex Get_Vertex_With_Operator_Id(operator_id i, vertex_list vl)
{
  with(vl)
  ( VertexListNull: AVertexNull,
    VertexListPair(av, tail):
    with (av)
    ( AVertexNull:Get_Vertex_With_Operator_Id(i, tail),
      AVertex(oi, *):
      with (oi)
      ( OperatorIdNull:Get_Vertex_With_Operator_Id(i, tail),
        OperatorId(*, *, *):
                (EqualOpId(i, oi)
                ? av
                : Get_Vertex_With_Operator_Id(i, tail))
      )
    )
  )

};
/*-------------------------------------------------------------------------------*/
BOOL Operator_Id_Not_In_Vertex_List(operator_id i, vertex_list vl)
       (with (vl)
        (      VertexListNull: true,
               VertexListPair(av, tail):
               with (av)
               (       AVertexNull:    Operator_Id_Not_In_Vertex_List(i, tail),
                       AVertex(oi, *):
                       with (oi)
                       (       OperatorIdNull: Operator_Id_Not_In_Vertex_List(i,
tail),
                               OperatorId(*, *, *):
                (EqualOpId(i, oi)
                ? false
                :Operator_Id_Not_In_Vertex_List(i, tail))
                       )
               )
        )
       );

/*-------------------------------------------------------------------------------*/
```

```
BOOL Id_Not_In_TypeImpl_Vertices(id i, type_impl ti)
{ with (ti)
  ( TypeImplNull:true,
    AdaTypeImpl(*):true,
    TypeImpl(*, oil):Id_Not_In_OpImplList(i, oil)
  )

};
/*-------------------------------------------------------------------------------*/
BOOL Id_Not_In_OpImplList(id i, operator_impl_list oil)
{ with (oil)
  ( OpImplListNull:true,
    OpImplListPair(toi, tail):
    with(toi)
    ( TOpImplNull:true,
      TOpImpl(*, o_imp):
            ((Operator_Id_Not_In_Vertex_List(
                OperatorId(
                     OptionalTypeIdNull,
                     i,
                     OperatorIdPairsNull),
                Get_Vertex_List(o_imp)))
            ? Id_Not_In_OpImplList(i, tail)
            : false)
    )
  )
);
/*-------------------------------------------------------------------------------*/
STR exported Get_Vertex_Type_Id_Name(a_vertex a)
{
         with(a)
                (AVertexNull: "",
                 AVertex(oid, *):
                 with(oid)
                                 (OperatorIdNull: "",
                                  OperatorId(ti, *, *):
          with (ti)
          (  OptionalTypeIdNull: "",
             OptionalTypeIdPrompt: "",
             OptionalTypeId(i): Get_Id(i)
          )
                                 )
                )
);
/*-------------------------------------------------------------------------------*/
STR exported Get_Vertex_Operator_Id_Name(a_vertex a)
{
         with(a)
                (AVertexNull: "",
                 AVertex(oid, *):
                 with(oid)
                                 (OperatorIdNull: "",
                                  OperatorId(*, i, *): Get_Id(i)
                                 )
                )
);

/*-------------------------------------------------------------------------------*/
operator_id_pairs exported Get_Vertex_OpIdPairs(a_vertex a)
{
         with(a)
                (AVertexNull: OperatorIdPairsNull,
                 AVertex(oid, *):
```

```
                with(oid)
                                (OperatorIdNull: OperatorIdPairsNull,
                                 OperatorId(*, *, op): op
        )
    )
);
/*-------------------------------------------------------------------------*/
STR exported Get_Operator_Id_Name(operator_id a)
{
        with(a)
               (OperatorIdNull: "",
                OperatorId(*, i, *): Get_Id(i)
               )
);

/*-------------------------------------------------------------------------*/
STR exported Get_Edge_From_Vertex_Type_Id_Name(from_vertex_id a)
{
        with(a)
               (FVertexIdNull: "",
                FVertexId(t, *, *):
     with (t)
     ( OptionalTypeIdNull:"",
       OptionalTypeIdPrompt:"",
       OptionalTypeId(i):Get_Id(i)
     )
               )
);

/*-------------------------------------------------------------------------*/
STR exported Get_Edge_From_Vertex_Operator_Id_Name(from_vertex_id a)
{
        with(a)
               (FVertexIdNull: "",
                FVertexId(*, i, *): Get_Id(i)
               )
);

/*-------------------------------------------------------------------------*/
operator_id_pairs exported Get_Edge_From_Vertex_OpIdPairs(from_vertex_id a)
{
        with(a)
               (FVertexIdNull: OperatorIdPairsNull,
                FVertexId(*, *, oip): oip
               )
);

/*-------------------------------------------------------------------------*/
STR exported Get_Edge_To_Vertex_Type_Id_Name(to_vertex_id a)
{
        with(a)
               (TVertexIdNull: "",
                TVertexId(t, *, *):
     with (t)
     ( OptionalTypeIdNull:"",
       OptionalTypeIdPrompt:"",
       OptionalTypeId(i):Get_Id(i)
     )
               )
);

/*-------------------------------------------------------------------------*/
STR exported Get_Edge_To_Vertex_Operator_Id_Name(to_vertex_id a)
{
        with(a)
               (TVertexIdNull: "",
                TVertexId(*, i, *): Get_Id(i)
               )
);

/*-------------------------------------------------------------------------*/
operator_id_pairs exported Get_Edge_To_Vertex_OpIdPairs(to_vertex_id a)
{
        with(a)
               (TVertexIdNull: OperatorIdPairsNull,
                TVertexId(*, *, oip): oip
               )
);

/*-------------------------------------------------------------------------*/
STR exported Get_Edge_Name(an_edge a)
{
        with(a)
               (AnEdgeNull: "",
                AnEdge(i, *, *, *): Get_Id(i)
               )
);
/*-------------------------------------------------------------------------*/
from_vertex_id exported Get_Edge_From_Vertex_Op_Id(an_edge a)
{
        with(a)
               (AnEdgeNull: FVertexIdNull,
                AnEdge(*, *, fid, *): fid
               )
);
/*-------------------------------------------------------------------------*/
to_vertex_id exported Get_Edge_To_Vertex_Op_Id(an_edge a)
{
        with(a)
               (AnEdgeNull: TVertexIdNull,
                AnEdge(*, *, *, tid): tid
               )
);
/*-------------------------------------------------------------------------*/
time exported Get_Vertex_Time(a_vertex v)
{
        with(v)
        (AVertexNull: TimeNull,
        AVertex(*, ot): Get_Time_From_Optional_Time(ot)
   )
);
/*-------------------------------------------------------------------------*/
time exported Get_Time_From_Optional_Time(optional_time ot)
{
  with (ot)
  (
    OptionalTimeNull: TimeNull,
    OptionalTimePrompt: TimeNull,
    OptionalTime(t): t
  )
);
/*-------------------------------------------------------------------------*/
time exported Get_Time_From_Latency(latency_time lt)
{
  with (lt)
  (
```

```
    LatencyTimeNull: TimeNull,
    LatencyTimePrompt: TimeNull,
    LatencyTime(t): t
  )
};
/*-------------------------------------------------------------------------*/
time exported Get_Edge_Time(an_edge a)
{
        with(a)
        (AnEdgeNull: TimeNull,
        AnEdge(*, lt, *, *): Get_Time_From_Latency(lt)
   )
};
/*-------------------------------------------------------------------------*/
INT exported Get_Time_Value(time t)
{
        with (t)
                (TimeNull: 0,
                 Time(tv, *):
                        with(tv)
                                (IntegerNull : 0,
                                 IntegerVal(digits) : STRtoINT(digits)
                                )
                 )
};
/*-------------------------------------------------------------------------*/
INT exported Convert_Time_To_Integer(time t)
{
        with (t)
        (TimeNull:        0,
                Time(tv, tu):
    with(tv)
    ( IntegerNull:0,
      IntegerVal(digits):
                with(tu)
                (UnitNil:STRtoINT(digits),
                 UnitMICROSECONDS: (STRtoINT(digits) / 100),
                 UnitMS:         STRtoINT(digits),
                 UnitSEC:        (STRtoINT(digits) * 1000),
                 UnitMIN:        (STRtoINT(digits) * 60000),
                 UnitHOURS: (STRtoINT(digits) * 3600000)
      )
                )
        )
};

/*-------------------------------------------------------------------------*/
INT exported Get_Time_Unit(time t)
{
        with (t)
        (       TimeNull:        0,
                Time(*, tu):
                with(tu)
                (       UnitNil:        0,
                        UnitMICROSECONDS: 1,
                        UnitMS:         2,
                        UnitSEC:        3,
                        UnitMIN:        4,
                        UnitHOURS: 5
                )
        )
};
```

```
/*-------------------------------------------------------------------------*/
vertex_list exported Get_Vertex_List_From_Operator_Impl_List(operator_impl_list oil)
{
  with (oil)
        (       OpImplListNull:                 VertexListNull,
                OpImplListPair(toi, tail):
                with (toi)
                (       TOpImplNull:
Get_Vertex_List_From_Operator_Impl_List(tail),
                        TOpImpl(*, oi):         Concat_Vertex_List(
                                                        Get_Vertex_List(oi),

Get_Vertex_List_From_Operator_Impl_List(tail))
                )
        )
};

/*-------------------------------------------------------------------------*/
vertex_list exported Get_Vertex_List(operator_impl p)
{
  with (p)
  ( OpImplNull: VertexListNull,
    AdaOpImpl(i): VertexListNull,
    OperatorImpl(g, *, *):
    with (g)
    ( GraphNull: VertexListNull,
      Graph(vl, *): vl
    )
  )
};
/*-------------------------------------------------------------------------*/
vertex_list exported Rest_Of_Vertex_List(vertex_list p)
{
  with (p)
  ( VertexListNull: VertexListNull,
    VertexListPair(*, vl): vl
  )
};
/*-------------------------------------------------------------------------*/
a_vertex exported Get_Vertex(vertex_list p)
{
  with (p)
  ( VertexListNull: AVertexNull,
    VertexListPair(v, *): v
  )
};
/*-------------------------------------------------------------------------*/
edge_list exported Get_Edge_List(operator_impl p)
{
  with (p)
  ( OpImplNull: EdgeListNil,
    AdaOpImpl(i): EdgeListNil,
    OperatorImpl(g, *, *):
    with (g)
    ( GraphNull: EdgeListNil,
      Graph(*, el): el
    )
  )
};
/*-------------------------------------------------------------------------*/
edge_list exported Rest_Of_Edge_List(edge_list p)
{
  with (p)
```

```
  ( EdgeListNil: EdgeListNil,
    EdgeListPair(*, el): el
  )
};
/*-------------------------------------------------------------------------*/
an_edge exported Get_Edge(edge_list p)
{
  with (p)
  ( EdgeListNil: AnEdgeNull,
    EdgeListPair(e, *): e
  )
};
/*-------------------------------------------------------------------------*/
o_inputs_list exported Get_Inputs_List(operator_spec p)
{
  with (p)
  (
    OperatorSpec(*, oil, *, *, *, *, *, *, *): oil
  )
};
/*-------------------------------------------------------------------------*/
o_outputs_list exported Get_Outputs_List(operator_spec p)
{
  with (p)
  (
    OperatorSpec(*, *, ool, *, *, *, *, *, *): ool
  )
};
/*-------------------------------------------------------------------------*/
o_states_list exported Get_State_List(operator_spec p)
{
  with (p)
  (
    OperatorSpec(*, *, *, osl, *, *, *, *, *): osl
  )
};
/*-------------------------------------------------------------------------*/
o_states_list exported Rest_Of_State_List(o_states_list p)
{
  with (p)
  ( StatesListNone: StatesListNone,
    StatesListPair(*, osl): osl
  )
};
/*-------------------------------------------------------------------------*/
o_states exported Get_State(o_states_list p)
{
  with (p)
  ( StatesListNone: OpStatesNone,
    StatesListPair(os, *): os
  )
};
/*-------------------------------------------------------------------------*/
STR exported Get_State_Name(o_states s)
{
  with (s)
  ( OpStatesNone: "",
    OpStates(td, *, *):
    with (td)
    ( TypeDeclNil: "",
      TypeDeclPair(ad, *):
      with (ad)
      ( ADeclNil: "",
        ADecl(il, *):
        with (il)
        (   IdNil: "",
            IdPair(i, *): Get_Id(i)
        )
      )
    )
  )
};
/*-------------------------------------------------------------------------*/
STR exported Get_Id(id i)
{
        with(i)
                (IdNull: "",
                 Id(x):  x)
};
/*-------------------------------------------------------------------------*/
STR exported Get_Decl_Type_Name(decl_type_name dtn)
{
  with (dtn)
  ( DTypeNameNull: "",
    DTypeInteger: "Integer",
    DTypeReal: "Real",
    DTypeBoolean: "Boolean",
/*
    DTypeException: "Exception",
*/
    DTypeSimpleId(i): Get_Id(i),
    DTypeUserDefined(i, *): Get_Id(i)
  )
};
/*-------------------------------------------------------------------------*/
graph exported Return_Graph(graph G)
{ with (G)
       (GraphNull: GraphNull,
        Graph(VertexListNull, *): GraphNull,
        Graph(*, *): G
       )
};
/*-------------------------------------------------------------------------*/
graph exported Get_Graph_From_Op_Impl(operator_impl p)
{ with (p)
  (OpImplNull: GraphNull,
   AdaOpImpl(*): GraphNull,
   OperatorImpl(g, *, *): g
  )
};
/*-------------------------------------------------------------------------*/
BOOL exported Empty_Graph(operator_impl p)
{ with (p)
  (OpImplNull: true,
   AdaOpImpl(*): true,
   OperatorImpl(g, *, *):
   with (g)
   (GraphNull:true,
    Graph(vl, *):
    with (vl)
    ( VertexListNull:true,
      VertexListPair(*,*): false
    )
   )
  )
};
```

```
/*-------------------------------------------------------------------------------*/
type_declarations exported Get_Stream_decl_From_Op_Impl(operator_impl p)
{ with (p)
  (OpImplNull: TypeDeclNil,
   AdaOpImpl(*): TypeDeclNil,
   OperatorImpl(*, d, *):
   with (d)
    ( /* DeclarationsNull:  TypeDeclNil, */
     Declarations(os, *):
     with (os)
      (StreamsNull: TypeDeclNil,
       StreamsPrompt: TypeDeclNil,
       Streams(td): td
      )
    )
  )
};

/*-------------------------------------------------------------------------------*/
/* the following routine is commented out because it is no longer being used
operator_id exported Get_Operator_Id_From_AConstraint(a_constraint ac)
{ with (ac)
  (AConstraintNull: IdNull,
   AConstraint(i, *, *, *, *, *, *, *, *): i
  )
};
*/
/*-------------------------------------------------------------------------------*/
/*
declarations exported Make_Null_Declarations()
{ DeclarationsNull
};
*/
/*-------------------------------------------------------------------------------*/
BOOL exported PSDL_Decomposition(operator_impl p)
{ with(p)
  (OpImplNull: false,
   AdaOpImpl(*): false,
   OperatorImpl(*, *, *): true
  )
};
/*-------------------------------------------------------------------------------*/
BOOL exported Null_Graph(graph g)
{ with(g)
       (GraphNull: true,
        Graph(VertexListNull, *): true,
        Graph(*, *): false
       )
};
/*-------------------------------------------------------------------------------*/
/*
BOOL exported Null_Declaration(declarations d)
{ with(d)
       (DeclarationsNull: true,
        Declarations(*, *): false
       )
};
*/
/*-------------------------------------------------------------------------------*/
decl_type_name Find_Type_Name(id i, type_declarations t)
{ with (t)
  (TypeDeclNil: DTypeNameNull,
   TypeDeclPair(ad, td):
   with(ad)
    (ADeclNil: Find_Type_Name(i, td),
     ADecl(idl, dtn): Id_In_IdList(i, idl)?dtn:Find_Type_Name(i, td)
    )
  )
};
/*-------------------------------------------------------------------------------*/
BOOL Id_In_IdList(id i, id_list idl)
{ with(idl)
  (IdNil: false,
   IdPair(idl_head, idl_tail): (i==idl_head)?true:Id_In_IdList(i, idl_tail)
  )
};
/*-------------------------------------------------------------------------------*/
component exported Dummy_Op()
{
    Op(
      Id("new_op"),
      OperatorSpec(
        GenericsListNone,
        InputsListNone,
        OutputsListNone,
        StatesListNone,
        ExcListNone,
        OpTimingInfoNone,
        KeyWordsNone,
        InformalDescsNull,
        FormalDescsNone),
      OpImplNull)
};

/*-------------------------------------------------------------------------------*/
optional_streams exported Build_Streams(IdSet ids, type_declarations td)
       (
    Streams(Build_Type_Decl(ids, td))
       );

/*-------------------------------------------------------------------------------*/
type_declarations exported Build_Type_Decl(IdSet ids, type_declarations td)
       (with(ids)
        (      IdSetNil:              TypeDeclNil,
               IdSetPair(i, tail):    ADecl(IdPair(i, IdNil), Find_Type_Name(i, td))
                                      :: Build_Type_Decl(tail, td)
        )
       );
/*-------------------------------------------------------------------------------*/
constraints exported Build_Constraints(OpIdSet ids, constraints cs)
       (with(ids)
        (      OpIdSetNil:              ConstraintsNull,
               OpIdSetPair(i, tail):    Build_A_Constraint(i, cs)
                                      :: Build_Constraints(tail, cs)
        )
       );

/*-------------------------------------------------------------------------------*/
a_constraint Build_A_Constraint(operator_id i, constraints cs)
  (with(cs)
   (ConstraintsNull:AConstraint(i,
                    OptionalTriggerNull,
                    OptPeriodNull,
                    OptFinishWithinNull,
                    OptMcpNull,
```

```
                    OptMrtNull,
                    OutputGuardsNil,
                    ExceptionOpsNull,
                    TimerOperationsNil),
    ConstraintsPair(ac, tail):
    with(ac)
    ( AConstraintNull:Build_A_Constraint(i, tail),
      AConstraint(a_id, *, *, *, *, *, *, *, *):
                (EqualOpId(i, a_id)
                ? ac
                : Build_A_Constraint(i, tail))
    )
   )
  );

/*-------------------------------------------------------------------------------*/
component exported Find_Component(id i, psdl_components pc)
  (with(pc)
   (PsdlNil:NoComponent,
    PsdlPair(c, tail):
    with(c)
    (
      NoComponent:Find_Component(i, tail),
      Data(c_id, *, *):
            ((c_id == i)?c:Find_Component(i, tail)),
      Op(o_id, *, *):
            ((o_id == i)?c:Find_Component(i, tail))
    )
   )
  );

/*-------------------------------------------------------------------------------*/
t_op_impl exported Find_TOpImpl_in_operator_impl_list(id i, operator_impl_list ol)
  (with(ol)
   (OpImplListNull:TOpImplNull,
    OpImplListPair(t, tail):
    with(t)
    (
      TOpImplNull:Find_TOpImpl_in_operator_impl_list(i, tail),
      TOpImpl(t_id, *):
            ((t_id == i)?t:Find_TOpImpl_in_operator_impl_list(i, tail)),
    )
   )
  );
/*-------------------------------------------------------------------------------*/
type_declarations exported Find_O_Inputs(id i, o_inputs_list il)
  (with(il)
   (InputsListNone:TypeDeclNil,
    InputsListPair(os, tail):
    with(os)
    ( OpInputsNone:Find_O_Inputs(i, tail),
      OpInputs(td, *):
                ((IsTypeDeclNil(Find_Type_Decl(i, td)))
                ? Find_O_Inputs(i, tail)
                : Find_Type_Decl(i, td))
    )
   )
  );

/*-------------------------------------------------------------------------------*/
type_declarations exported Find_O_Outputs(id i, o_outputs_list ol)
  (with(ol)
   (OutputsListNone:TypeDeclNil,
    OutputsListPair(os, tail):
    with(os)
    ( OpOutputsNone:Find_O_Outputs(i, tail),
      OpOutputs(td, *):
                ((IsTypeDeclNil(Find_Type_Decl(i, td)))
                ? Find_O_Outputs(i, tail)
                : Find_Type_Decl(i, td))
    )
   )
  );

/*-------------------------------------------------------------------------------*/
type_declarations exported Find_Type_Decl(id i, type_declarations td)
  (with(td)
   (TypeDeclNil:TypeDeclNil,
    TypeDeclPair(ad, tail):
    with(ad)
    ( ADeclNil:Find_Type_Decl(i, tail),
      ADecl(idl, *):
            ((Id_In_IdList(i, idl))
            ? td
            : Find_Type_Decl(i, tail))
    )
   )
  );

/*-------------------------------------------------------------------------------*/
type_declarations exported Find_Stream_Type_Decl(id i, optional_streams st)
  (with(st)
   (StreamsNull:TypeDeclNil,
    StreamsPrompt:TypeDeclNil,
    Streams(td):Find_Type_Decl(i, td)
   )
  );

/*-------------------------------------------------------------------------------*/
a_constraint exported Find_A_Constraint(operator_id i, constraints ct)
  (with(ct)
   (ConstraintsNull:AConstraintNull,
    ConstraintsPair(ac, tail):
    with(ac)
    ( AConstraintNull:Find_A_Constraint(i, tail),
      AConstraint(a_id, *,*,*,*,*,*,*,*):
                (EqualOpId(a_id, i)
                ? ac
                :Find_A_Constraint(i, tail))
    )
   )
  );
/*-------------------------------------------------------------------------------*/
o_inputs_list exported Make_Inputs_List(type_declarations td)
  (with(td)
   (TypeDeclNil:InputsListNone,
    TypeDeclPair(*, *):InputsListPair(
                OpInputs(td, ReqmtsTraceNone),
                InputsListNone)


/*  the following code separates the id_lists into individual input_lists */
/*  not use in current version of psdl_editor */
/*
    TypeDeclPair(ad, tail):InputsListPair(
                OpInputs(
```

```
                    TypeDeclPair(
                                ad,
                                TypeDeclNil),
                    ReqmtsTraceNone),
                Make_Inputs_List(tail))
*/
   )
  );


/*-------------------------------------------------------------------------------*/
o_outputs_list exported Make_Outputs_List(type_declarations td)
  (with(td)
   (TypeDeclNil:OutputsListNone,
    TypeDeclPair(ad, tail):OutputsListPair(
                OpOutputs(td, ReqmtsTraceNone),
                OutputsListNone)

/* the following code separates the id_lists into individual output_lists */
/*  not use in current version of psdl_editor */
/*
    TypeDeclPair(ad, tail):OutputsListPair(
                OpOutputs(
                    TypeDeclPair(
                                ad,
                                TypeDeclNil),
                    ReqmtsTraceNone),
                Make_Outputs_List(tail))
*/
   )
  );


/*-------------------------------------------------------------------------------*/
operator_spec exported Replace_Input_Output_Met(operator_spec os,
                o_inputs_list new_il,
                o_outputs_list new_ol,
                o_timing_info new_met)
  (with(os)
   (OperatorSpec(gl, il, ol, sl, el, ti, k, id, fd):
      OperatorSpec(gl, new_il, new_ol, sl, el, new_met, k, id, fd)
   )
  );

/*-------------------------------------------------------------------------------*/
operator_impl exported Replace_Stream_Constraint_List(operator_impl oi,
                optional_streams new_stream_list,
                constraints new_constraint_list)
  (with(oi)
   (OperatorImpl(g, d, c):
      OperatorImpl(g, Make_Declarations(d, new_stream_list),
            Make_Cc(c, new_constraint_list))
   )
  );

/*-------------------------------------------------------------------------------*/
type_declarations exported Concat_Type_Decl_List(type_declarations td1,
type_declarations td2)
        (with(td1)
                (TypeDeclNil:                   td2,
                 TypeDeclPair(ad, tail):        ad::Concat_Type_Decl_List(tail, td2)
                )
        );
/*-------------------------------------------------------------------------------*/
component exported Make_Op(id i, operator_spec os, operator_impl oi)
  {
    Op(i, os, oi)
  };
/*-------------------------------------------------------------------------------*/
operator_impl exported Make_OpImplNull()
  {
    OpImplNull
  };
/*-------------------------------------------------------------------------------*/
t_op_impl exported Make_TOpImpl(id i, operator_impl oi)
  {
    TOpImpl(i, oi)
  };
/*-------------------------------------------------------------------------------*/
t_op_impl exported Make_TOpImplNull()
  {
    TOpImplNull()
  };
/*-------------------------------------------------------------------------------*/
declarations exported Make_Empty_Declarations()
  {
    Declarations(StreamsNull, TimersNull)
  };
/*-------------------------------------------------------------------------------*/
/*
declarations exported Make_DeclarationsNull()
  {
    DeclarationsNull
  };
*/
/*-------------------------------------------------------------------------------*/
cc exported Make_CcNull()
  {
    CcNull
  };
/*-------------------------------------------------------------------------------*/
cc exported Make_Cc(cc old_cc, constraints new_cs)
  (with(old_cc)
   (CcNull:
    with(new_cs)
    ( ConstraintsNull:CcNull,
      ConstraintsPair(*,*):Cc(new_cs, InformalDescsNull)
    ),
    Cc(*, des):
    with(new_cs)
    ( ConstraintsNull:
      with(des)
      ( InformalDescsNull:CcNull,
        InformalDescsPrompt:CcNull,
        InformalDescs(*):Cc(new_cs, des)
      ),

      ConstraintsPair(*,*):Cc(new_cs, des)
    )
   )
  );
/*-------------------------------------------------------------------------------*/
declarations exported Make_Declarations(declarations old_d, optional_streams new_s)
  (with(old_d)
   (/*
```

```
    DeclarationsNull:
    with(new_s)
    ( StreamsNull:DeclarationsNull,
      StreamsPrompt:DeclarationsNull,
      Streams(*):Declarations(new_s, TimersNull)
    ),

    Declarations(*, ot):
    with(new_s)
    ( StreamsNull:
      with(ot)
      ( TimersNull:DeclarationsNull,
        TimersPrompt:DeclarationsNull,
        Timers(*):Declarations(new_s, ot)
      ),

      StreamsPrompt:Declarations(new_s, ot),

      Streams(*):Declarations(new_s, ot)
    )
    */

    Declarations(*, ot):Declarations(new_s, ot)
   )
  };
/*-------------------------------------------------------------------------*/
o_timing_info exported Make_New_O_Timing_Info(time new_time, o_timing_info old_info)
{
  with (new_time)
  (TimeNull:OpTimingInfoNone,
  Time(*,*):
  with(old_info)
  ( OpTimingInfoNone:OpTimingInfo(new_time, ReqmtsTraceNone),
    OpTimingInfoPrompt:OpTimingInfo(new_time, ReqmtsTraceNone),
    OpTimingInfo(*, req):OpTimingInfo(new_time, req)
  )
  )
};
/*-------------------------------------------------------------------------*/
psdl_components exported Concat_Psdl_Components(psdl_components o1, psdl_components o2)
{
  with(o1)
  (PsdlNil:o2,
  PsdlPair(c, tail):c::Concat_Psdl_Components(tail, o2)
  )
};

/*-------------------------------------------------------------------------*/
psdl_components exported Make_New_Ops(IdSet op_set)
{
  with (op_set)
  (IdSetNil:PsdlNil,
  IdSetPair(i, tail):Op(i,
            OperatorSpec(
                GenericsListNone,
                InputsListNone,
                OutputsListNone,
                StatesListNone,
                ExcListNone,
                OpTimingInfoNone,
                KeyWordsNone,
                InformalDescsNull,
                FormalDescsNone),
                OpImplNull)
        :: Make_New_Ops(tail)


  )
};

/*-------------------------------------------------------------------------*/
component exported Make_New_Op_Component(id i, psdl_components o, type_declarations td)
{
  with (i)
  (IdNull:NoComponent,
  Id(*):
      Op(i,
        OperatorSpec(
            GenericsListNone,
/*
            InputsListNone,
            OutputsListNone,
*/
            Build_InputsList(i, o, td),
            Build_OutputsList(i, o, td),
            StatesListNone,
            ExcListNone,
/*
            OpTimingInfoNone,
*/
            Build_Met(
                OperatorId(
                    OptionalTypeIdNull,
                    i,
                    OperatorIdPairsNull),
                o),
            KeyWordsNone,
            InformalDescsNull,
            FormalDescsNone),
        OpImplNull)
  )
};

/*-------------------------------------------------------------------------*/
/*

psdl_components exported Make_Psdl_Pair_With_Single_Component(component c)
{
  PsdlPair(c, PsdlNil)
};
*/

/*-------------------------------------------------------------------------*/
/*
psdl_components exported Make_Psdl_Nil()
{
  PsdlNil
};
*/

/*-------------------------------------------------------------------------*/
operator_impl_list exported Make_New_Op_Impl_List(IdSet op_set)
{
  with (op_set)
  (IdSetNil:OpImplListNull,
  IdSetPair(i, tail):OpImplListPair(
```

```
            TOpImpl(i, OpImplNull),
            Make_New_Op_Impl_List(tail))
  )
};

/*-------------------------------------------------------------------------*/
operator_impl_list exported Concat_Op_Impl_List(
      operator_impl_list oil_1, operator_impl_list oil_2)
{
  with (oil_1)
  (OpImplListNull:oil_2,
  OpImplListPair(toi, tail):OpImplListPair(
                toi,
                Concat_Op_Impl_List(tail, oil_2))
  )
};

/*-------------------------------------------------------------------------*/
type_impl exported Make_TypeImpl(operator_impl_list oil)
{
  TypeImpl(TypeNameNull, oil)
};
/*-------------------------------------------------------------------------*/

IdSet Extract_From_Edge_Ids(id i, edge_list el)
        (with (el)
         (      EdgeListNil:                    IdSetNil,
                EdgeListPair(ae, tail):
                with (ae)
                (       AnEdgeNull:             Extract_From_Edge_Ids(i, tail),
                        AnEdge(ei, lt, fv, tv):
                        with (fv)
                        (       FVertexIdNull:  Extract_From_Edge_Ids(i, tail),
                                FVertexId(*, f_id, *):
                                                ((i == f_id)
                                                ? IdSetUnion(
                                                        SingletonIdSet(ei),
                                                        Extract_From_Edge_Ids(i, tail))
                                                : Extract_From_Edge_Ids(i, tail))
                        )
                )

         )
        );

/*-------------------------------------------------------------------------*/
IdSet Extract_To_Edge_Ids(id i, edge_list el)
        (with (el)
         (      EdgeListNil:                    IdSetNil,
                EdgeListPair(ae, tail):
                with (ae)
                (       AnEdgeNull:             Extract_To_Edge_Ids(i, tail),
                        AnEdge(ei, lt, fv, tv):
                        with(tv)
                        (       TVertexIdNull:  Extract_To_Edge_Ids(i, tail),
                                TVertexId(*, t_id, *):
                                                ((i == t_id)
                                                ? IdSetUnion(
                                                        SingletonIdSet(ei),
                                                        Extract_To_Edge_Ids(i, tail))
                                                : Extract_To_Edge_Ids(i, tail))
                        )
                )
         )
        );

/*-------------------------------------------------------------------------*/
IdSet exported Extract_Input_Id_Set(id i, psdl_components o)
        (with (o)
         (      PsdlNil:IdSetNil,
                PsdlPair(c, tail):
                with (c)
                (       NoComponent:    Extract_Input_Id_Set(i, tail),
                        Data(*, *, ti):((Id_Not_In_TypeImpl_Vertices(i, ti))
                 ? Extract_Input_Id_Set(i, tail)
                 : Extract_Input_Id_Set_From_TypeImpl(i, ti)),

                        Op(*, *, o_imp):        ((Operator_Id_Not_In_Vertex_List(
                    OperatorId(
                                OptionalTypeIdNull,
                                i,
                                OperatorIdPairsNull),
                    Get_Vertex_List(o_imp)))
                                                ? Extract_Input_Id_Set(i, tail)
                                                : Extract_To_Edge_Ids(i,
Get_Edge_List(o_imp)))
                )
         )
        );

/*-------------------------------------------------------------------------*/
IdSet exported Extract_Output_Id_Set(id i, psdl_components o)
        (with (o)
         (      PsdlNil:                        IdSetNil,
                PsdlPair(c, tail):
                with (c)
                (       NoComponent:            Extract_Output_Id_Set(i, tail),
                        Data(*, *, ti):((Id_Not_In_TypeImpl_Vertices(i, ti))
                 ? Extract_Output_Id_Set(i, tail)
                 : Extract_Output_Id_Set_From_TypeImpl(i, ti)),
                        Op(*, *, o_imp):        ((Operator_Id_Not_In_Vertex_List(
                    OperatorId(
                                OptionalTypeIdNull,
                                i,
                                OperatorIdPairsNull),
                    Get_Vertex_List(o_imp)))
                                                ? Extract_Output_Id_Set(i, tail)
                                                : Extract_From_Edge_Ids(i,
Get_Edge_List(o_imp)))
                )
         )
        );


/*-------------------------------------------------------------------------*/
IdSet exported Extract_Input_Id_Set_From_TypeImpl(id i, type_impl ti)
{ with (ti)
  ( TypeImplNull:IdSetNil,
    AdaTypeImpl(*):IdSetNil,
    TypeImpl(*, oil):Extract_Input_Id_Set_From_OpImplList(i, oil)
  )
};

/*-------------------------------------------------------------------------*/
IdSet exported Extract_Input_Id_Set_From_OpImplList(id i, operator_impl_list oil)
```

```
{ with (oil)
  { OpImplListNull:IdSetNil,
    OpImplListPair(toi, tail):
    with(toi)
    { TOpImplNull:IdSetNil,
      TOpImpl(*, o_imp):
            ((Operator_Id_Not_In_Vertex_List(
                OperatorId(
                    OptionalTypeIdNull,
                    i,
                    OperatorIdPairsNull),
                Get_Vertex_List(o_imp)))
            ? Extract_Input_Id_Set_From_OpImplList(i, tail)
            : Extract_To_Edge_Ids(i, Get_Edge_List(o_imp)))
    }
  }
};
/*-----------------------------------------------------------------------------*/
IdSet exported Extract_Output_Id_Set_From_TypeImpl(id i, type_impl ti)
{ with (ti)
  { TypeImplNull:IdSetNil,
    AdaTypeImpl(*):IdSetNil,
    TypeImpl(*, oil):Extract_Output_Id_Set_From_OpImplList(i, oil)
  }
};

/*-----------------------------------------------------------------------------*/
IdSet exported Extract_Output_Id_Set_From_OpImplList(id i, operator_impl_list oil)
{ with (oil)
  { OpImplListNull:IdSetNil,
    OpImplListPair(toi, tail):
    with(toi)
    { TOpImplNull:IdSetNil,
      TOpImpl(*, o_imp):
            ((Operator_Id_Not_In_Vertex_List(
                OperatorId(
                    OptionalTypeIdNull,
                    i,
                    OperatorIdPairsNull),
                Get_Vertex_List(o_imp)))
            ? Extract_Output_Id_Set_From_OpImplList(i, tail)
            : Extract_From_Edge_Ids(i, Get_Edge_List(o_imp)))
    }
  }
};
/*-----------------------------------------------------------------------------*/
o_inputs_list exported Make_InputsListPair_From_Type_Decl(type_declarations td)
  {
    InputsListPair(
      OpInputs(td, ReqmtsTraceNone),
      InputsListNone)
  };


/*-----------------------------------------------------------------------------*/
o_inputs_list exported Make_InputsListNone()
  {
    InputsListNone
  };


/*-----------------------------------------------------------------------------*/
o_outputs_list exported Make_OutputsListPair_From_Type_Decl(type_declarations td)
  {
    OutputsListPair(
      OpOutputs(td, ReqmtsTraceNone),
      OutputsListNone)
  };


/*-----------------------------------------------------------------------------*/
o_outputs_list exported Make_OutputsListNone()
  {
    OutputsListNone
  };


/*-----------------------------------------------------------------------------*/
o_timing_info exported Make_OpTimingInfo_From_Met(time t)
  {
    OpTimingInfo(t, ReqmtsTraceNone)
  };


/*-----------------------------------------------------------------------------*/
o_timing_info exported Make_OpTimingInfoNone()
  {
    OpTimingInfoNone
  };

/*-----------------------------------------------------------------------------*/
component exported Make_Op_From_SSLstring(STR name)
  {
    Op(Id(name),
                OperatorSpec(
                                GenericsListNone,
                                InputsListNone,
                                OutputsListNone,
                                StatesListNone,
                                ExcListNone,
                                OpTimingInfoNone,
                                KeyWordsNone,
                                InformalDescsNull,
                                FormalDescsNone),
      OperatorImpl(
         GraphNull,
         /*
         DeclarationsNull,
         */
         Declarations(StreamsNull, TimersNull),
         CcNull)
    )
  };

/*-----------------------------------------------------------------------------*/
/* the following routine is commented out because it is no longer being used
BOOL exported Return_False()
{
  false
};
*/

/*-----------------------------------------------------------------------------*/
psdl_components exported Merge_Psdl_Components(psdl_components pc_1, psdl_components
pc_2)
{
```

```
  with (pc_1)
  ( PsdlNil:     pc_2,
    PsdlPair(h_1, t_1):
    with(h_1)
    ( NoComponent:Merge_Psdl_Components(t_1, pc_2),
      Data(head_id_1, *, *):
      with (pc_2)
      ( PsdlNil: pc_1,
        PsdlPair(h_2, t_2):
        with (h_2)
        (    NoComponent:Merge_Psdl_Components(pc_1, t_2),
             Data(head_id_2, *, *):
                      ((head_id_1 <= head_id_2)
                      ? h_1::Merge_Psdl_Components(t_1, pc_2)
                      : h_2::Merge_Psdl_Components(pc_1, t_2)),
             Op(*, *, *):h_1::Merge_Psdl_Components(t_1, pc_2)
        )
      ),
      Op(head_id_1, *, *):
      with (pc_2)
      ( PsdlNil: pc_1,
        PsdlPair(h_2, t_2):
        with (h_2)
        (    NoComponent:Merge_Psdl_Components(pc_1, t_2),
             Data(*, *, *):h_2::Merge_Psdl_Components(pc_1, t_2),
             Op(head_id_2, *, *):
                      ((head_id_1 <= head_id_2)
                      ? h_1::Merge_Psdl_Components(t_1, pc_2)
                      : h_2::Merge_Psdl_Components(pc_1, t_2))
        )
      )
    )
  )
);


/*-------------------------------------------------------------------------------*/
psdl_components exported Sort_Psdl_Components(psdl_components pc)
{
  with (pc)
  ( PsdlNil:PsdlNil,
    PsdlPair(head, tail):Insert_Component(
                head,
                Sort_Psdl_Components(tail)
                )
  )
};


/*-------------------------------------------------------------------------------*/
psdl_components exported Insert_Component(component h, psdl_components pc)
{
  with (pc)
  ( PsdlNil: PsdlPair(h, PsdlNil),
    PsdlPair(head, tail):
    with (head)
    ( NoComponent:Insert_Component(h, tail),
      Data(head_id, *, *):
      with (h)
      ( NoComponent:pc,
        Data(h_id, *, *):((h_id <= head_id)
                    ? h::pc
                    : head::Insert_Component(h, tail)),
          Op(*, *, *):head::Insert_Component(h, tail)
        ),
        Op(head_id, *, *):
        with(h)
        ( NoComponent:    pc,
          Data(h_id, *, *):h::pc,
          Op(h_id, *, *):((h_id <= head_id)
                      ? h::pc
                      : head::Insert_Component(h, tail))
        )
      )
    )
  )
);

/*-------------------------------------------------------------------------------*/
prototype exported Make_Proto(psdl_components pc)
{
  Prot(pc)
};

/*-------------------------------------------------------------------------------*/
operator_impl exported Make_Op_Impl(graph g, declarations d, cc c)
{
  OperatorImpl(g, d, c)
};

/*-------------------------------------------------------------------------------*/
operator_impl exported Make_AdaOpImpl(id i)
{
  AdaOpImpl(i)
};

/*-------------------------------------------------------------------------------*/
type_impl exported Make_AdaTypeImpl(id i)
{
  AdaTypeImpl(i)
};
/*-------------------------------------------------------------------------------*/
BOOL PSDLTypeImpl(type_impl ti)
{
  with (ti)
  (
    TypeImplNull:false,
    AdaTypeImpl(*):false,
    TypeImpl(*, *):true
  )
};

/*-------------------------------------------------------------------------------*/
/*
t_oper_spec Find_T_Op_Spec_in_Data(id i, component d)
{
  with (d)
  ( NoComponent:TOpSpecNil,
    Data(i, ts, ti):
    with (ts)
    ( TypeSpec(*,*,oo,*,*,*):Find_T_Op_Spec(i, oo)
    ),
    Op(*, *, *):TOpSpecNil
  )
};
*/
/*-------------------------------------------------------------------------------*/
```

```
t_oper_spec exported Find_T_Op_Spec(id i, o_operators oo)
{
  with (oo)
  ( OperatorNil:TOpSpecNil,
    OperatorPair(tos, tail):
    with (tos)
    ( TOpSpecNil:Find_T_Op_Spec(i, tail),
      TOpSpec(t_id, os):(i == t_id)
                ? tos
                : Find_T_Op_Spec(i, tail)
    )
  )
};

/*-------------------------------------------------------------------------*/
t_oper_spec exported Make_TOpSpecNil()
{
  TOpSpecNil
};
/*-------------------------------------------------------------------------*/
operator_impl Find_Operator_Impl(id i, operator_impl_list oil)
{
  with (oil)
  ( OpImplListNull:OpImplNull,
    OpImplListPair(toi, tail):
    with (toi)
    ( TOpImplNull:Find_Operator_Impl(i, tail),
      TOpImpl(t_id, oi):(i == t_id)
                ? oi
                : Find_Operator_Impl(i, tail)
    )
  )
};

/*-------------------------------------------------------------------------*/
IdSet Extract_Edge_Id_Set(operator_impl oi)
{
  with (oi)
  ( OpImplNull:IdSetNil,
    AdaOpImpl(*):IdSetNil,
    OperatorImpl(g, *, *):
    with (g)
    ( GraphNull:IdSetNil,
      Graph(*, el):Make_Edge_Id_Set(el)
    )
  )
};


/*-------------------------------------------------------------------------*/
IdSet Make_Edge_Id_Set(edge_list el)
{
  with (el)
  ( EdgeListNil:IdSetNil,
    EdgeListPair(ae, tail):
    with (ae)
    ( AnEdgeNull:Make_Edge_Id_Set(tail),
      AnEdge(e_i, *, *, *):
      with (e_i)
      ( IdNull:  Make_Edge_Id_Set(tail),
        Id(*):   IdSetUnion(
                     SingletonIdSet(e_i),
                     Make_Edge_Id_Set(tail))
      )
    )
  )
};


/*************************************************************************
****  This third set of functions were already existing and
****  for generating an operator head_node and its operator record.
****  Documented originally as file: ed.2.ssl
*************************************************************************/
OPNodePTR Make_Operator_Node(
      name, optional_type_name, oper_name,
      parameter_list, id, met, X, Y, radius, color,
      name_font, name_x, name_y,
      met_font, met_x, met_y,
      is_deleted, is_new, is_composite,
      is_terminator, is_modified)
char
   *name,
   *optional_type_name,
   *oper_name,
   *parameter_list;
OP_ID
   id;
int
   met,
   X,
   Y,
   radius,
   color,
   name_font,
   name_x,
   name_y,
   met_font,
   met_x,
   met_y;
BOOL
   is_deleted,
   is_new,
   is_composite,
   is_terminator,
   is_modified;
{
  OPNodePTR p;
  OPERATOR q;

#ifdef GRAPHICS_DEBUG
  /* debugging */
  printf("Entering Make_Operator_Node\n");
#endif

#ifdef SDE_DEBUG_3
printf(" Make_Operator_Node: name = %s\n", name);
printf(" Make_Operator_Node: optional_type_name = %s\n", optional_type_name);
printf(" Make_Operator_Node: oper_name = %s\n", oper_name);
printf(" Make_Operator_Node: parameter_list = %s\n", parameter_list);
#endif

  p = (OPNodePTR)malloc(sizeof(OP_HEAD));
  q = (OPERATOR)malloc(sizeof(OP_NODE));
  p->op = q;
  p->next = NULL;
```

```
  q->name = strdup(name);
  q->optional_type_name = strdup(optional_type_name);
  q->oper_name = strdup(oper_name);
  q->parameter_list = strdup(parameter_list);
  q->name_font = name_font;
  q->name_x = name_x,
  q->name_y = name_y,
  q->id = id;
  q->met = met;
  q->met_font = met_font;
  q->met_x = met_x;
  q->met_y = met_y;
  q->x = X;
  q->y = Y;
  q->radius = radius;
  q->color = color;
  q->is_deleted = is_deleted;
  q->is_new = is_new;
  q->is_composite = is_composite;
  q->is_terminator = is_terminator;
  q->is_modified = is_modified;


  return (p);
}
/*-----------------------------------------------------------------------------*/
TYPE_LIST Make_Type_Node(name)
  char *name;
{

  TYPE_LIST
  h;

  PROD_INSTANCE
  NullSet();

  h = (TYPE_LIST)malloc(sizeof(TYPE_NODE));
  h->type_name = strdup(name);
  h->undefined_op_impl = NullSet();
  h->obsolete_op_impl = NullSet();
  h->next = Global_Type_List;
  Global_Type_List = h;
  return(h);
}
/*-----------------------------------------------------------------------------*/
HeadPtr Make_Operator_Header(name, stream_list, operator_list, key, prod_no, del_flag)
  char
  *name;
  ST_PTR
  stream_list;
  OPNodePTR
  operator_list;
  int
  key,
  prod_no;


  {
        extern int op_id_count;

        HeadPtr
          h;
  PROD_INSTANCE
    NullSet();

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Make_Operator_Header\n");
/*
  printf("Make_Operator_Header: name = %s\n", name);
  printf("Make_Operator_Header: key = %d\n", key);
  printf("Make_Operator_Header: prod_no = %d\n", prod_no);
*/
#endif

        h = (HeadPtr)malloc(sizeof(HEADER_NODE));
        h->name = strdup(name);
        h->ada_op_name = strdup(name);
  h->global_id = strdup("0");
  h->local_id = strdup("0");
  h->type_id = strdup("");

#ifdef SDE_DEBUG_2
        printf("NAME INSERTED --> %s\n", name);
#endif

  h->parent = strdup("");
  h->parent_type_name = strdup("");
        h->stream_list = stream_list;
        h->operator_list = operator_list;
        h->op_id_no = key;
        h->prod_no = prod_no;
  h->met = fake_met;
  h->is_composite = fake_is_composite;
        h->marked_for_delete = del_flag;
        h->next = NULL;
  h->input_error = false;
  h->multi_op_error = false;
  h->output_error = false;
  h->met_error = false;
  h->stream_error = false;
  h->constraint_error = false;

        h->name_font = fake_font;
        h->name_x = fake_X;
        h->name_y = fake_Y;

        h->inh_input_id_set = NullSet();
        h->inh_output_id_set = NullSet();
        h->inh_met = NULLVALUE;
        h->state_id_set = NullSet();
        h->vertex_id_set = NullSet();
        h->edge_id_set = NullSet();
/*
        h->stream_id_set = NullSet();
  h->constraints_id_set = NullSet();
*/

        h->inh_input_decl = NULLVALUE;
        h->inh_output_decl = NULLVALUE;

        return(h);
  }
/*-----------------------------------------------------------------------------*/

/* converts integer to STR0 string structure */
```

```
FOREIGN C_INT_to_STR0(I)
int I;
{
  char buf[20];
  sprintf(buf, "%d", I);
  return(Str(str_to_str0(buf)));
}
/*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/
/* The following function uses "the_operator_list" and "the_stream_list"
   and builds the PSDL terms based on the "live" nodes.  The following can
   be built from the c-data structures:

   - Operator Specification (in case an operator is new)
   - Graph
   - Data stream declarations

   Only those fields represented in the data structure are relevant to
   this construction.  Optional phyla and fields whose values cannot be
   deduced from the data structures are represented by their NULL constructors.

   decls.h contains the necessary macros for building the terms. This file
   is generated by the SSL and erased after code generation.
*/
/*-------------------------------------------------------------------------*/
/*
  This function creates a graph with the new data stored in the c-data
   data structure.
*/

/*-------------------------------------------------------------------------*/
PROD_INSTANCE Make_Graph(o_list, s_list)
  OPNodePTR
  o_list;
  ST_PTR
  s_list;
{
  HeadPtr
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(),
        h;

  PROD_INSTANCE
  Make_Vertex_List(),
  Make_Edge_List(),
        graph,
        vertex_list,
        edge_list;
  boolean
  t_bool;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Make_Graph\n");
#endif

  /* Generate Vertex List */
  vertex_list = Make_Vertex_List(o_list);

  /* Generate Edge List */
  edge_list = Make_Edge_List(s_list);

  /* Generate the Graph */
  graph = Return_Graph(Graph(vertex_list, edge_list));


  /*debugging*/
/*
  printf("******* Make_Graph debugging **********\n");
  printf("%s\n", current_graph->name);
  Print_OperatorS_Operators(current_graph->operator_list);
  Print_OperatorS_Streams(current_graph->stream_list);

  printf("------- actual parameter --------------\n");
  Print_OperatorS_Operators(o_list);
  Print_OperatorS_Streams(s_list);
*/


  return(graph);
}
/*-------------------------------------------------------------------------*/
OPNodePTR Sort_Operator_List(q)
  OPNodePTR q;
{
  OPNodePTR
  head,
  tail;
  boolean
  done;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Sort_Operator_List\n");
#endif

  if (q == NULL)
  {
  return(q);
  }
  else
  {
  if (q->next == NULL)
  {
    return(q);
  }
  else
  {

    tail = Sort_Operator_List(q->next);
    if (strcmp(q->op->name, tail->op->name) <= 0)
    {
      q->next = tail;
      return(q);
    }
    else
    {
      head = tail;
      done = false;
      while (tail->next != NULL && !done)
      {
        if (strcmp(q->op->name, tail->next->op->name) <= 0)
        {
            done = true;
        }
        else
        {
            tail = tail->next;
```

```
        }
      }
      q->next = tail->next;
      tail->next = q;
      return(head);
    }
  }
  }
}

/*-------------------------------------------------------------------------*/
ST_PTR Sort_Stream_List(p)
  ST_PTR p;
{
  ST_PTR
  head,
  tail;
  boolean
  done;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Sort_Stream_List\n");
#endif

  if (p == NULL)
  {
  return(p);
  }
  else
  {
  if (p->next == NULL)
  {
    return(p);
  }
  else
  {

    tail = Sort_Stream_List(p->next);
    if (strcmp(p->st->name, tail->st->name) <= 0)
    {
      p->next = tail;
      return(p);
    }
    else
    {
      head = tail;
      done = false;
      while (tail->next != NULL && !done)
      {
        if (strcmp(p->st->name, tail->next->st->name) <= 0)
        {
            done = true;
        }
        else
        {
            tail = tail->next;
        }
      }
      p->next = tail->next;
      tail->next = p;
      return(head);
    }
  }
  }
}


/*-------------------------------------------------------------------------*/

/*
   This function "sweeps" through the list of operators and builds a list
   of vertices of the PSDL graph.  Only operators that are live are placed
   in the new PSDL vertex list.

   Note: The following code ASSUMES
   (1) AVertex(operator_id optional_time), OperatorId(id operator_id_pairs)
       where operator_id_pair always equal to OperatorIdPairsNull()

   (2) time units are always in MS.

*/
PROD_INSTANCE Make_Vertex_List(p)
         OPNodePTR p;

{ OPERATOR
        q;

  PROD_INSTANCE
/*
        name,
*/
  front_part,
  middle_part,
  suffix_part,
  make_operator_id_pairs(),
        time,
        vertex;

  int
  Get_Unique_Id();

  char
  *get_vertex_type_name(),
  *get_vertex_oper_name(),
  *get_vertex_parameters(),
  *optional_type_name,
  *oper_name,
  *parameter_list,
  *clean_name,
  *remove_blanks_from_string(),
  dummy_name[100];

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Make_Vertex_List\n");
#endif

  if (p == NULL)
        {
        return(VertexListNull);
  }
  else
        {q = p->op;   /* q points to the operator's record */

   if (strcmp(q->name, "") == 0)
```

```
  {  /* need to create dummy name */
   sprintf(dummy_name, "NONAME_%d", Get_Unique_Id());
   q->name = strdup(dummy_name);
  }
  else
  {
   clean_name = remove_blanks_from_string(q->name);
   if (strcmp(clean_name, q->name) != 0)
   {
     free(q->name);
     q->name = clean_name;
   }
   else
   {
     free(clean_name);
   }
}


      if (q->is_deleted != TRUE)  /* Make up the Vertex for this Operator */
        {

              if (q->is_new)
   {
              q->id = Get_Unique_Id();
    q->is_new = false;
    q->optional_type_name = get_vertex_type_name(q->name);
    q->oper_name = get_vertex_oper_name(q->name);
    q->parameter_list = get_vertex_parameters(q->name);

    if (strcmp(q->parameter_list, "") == 0)
    {
      if (strcmp(q->optional_type_name, "") != 0)
      {
          /* there is a syntax error in the parameter list */
          free(q->optional_type_name);
          q->optional_type_name = strdup("");
          free(q->name);
          q->name = strdup(q->oper_name);
      }

    }
    else
    {
      if (strcmp(q->optional_type_name, "") == 0)
      {
          /* missing optional_type_name */
          free(q->parameter_list);
          q->parameter_list = strdup("");
          free(q->name);
          q->name = strdup(q->oper_name);
      }
    }
   }
   else
   {
    sprintf(dummy_name, "%s%s%s%s%s%s", q->optional_type_name,
          ".", q->oper_name, "(", q->parameter_list, ")");

    if (strcmp(dummy_name, q->name) != 0)
    {
      /* vertex name differs from the original one */
      q->optional_type_name = get_vertex_type_name(q->name);
      q->oper_name = get_vertex_oper_name(q->name);
      q->parameter_list = get_vertex_parameters(q->name);
     }
     }


    /* create the necessary phylum for the operator */
     if (strcmp(q->optional_type_name, "") != 0)
      front_part = OptionalTypeId(Id(SSLstring(q->optional_type_name)));
     else
      front_part = OptionalTypeIdNull;


     if (strcmp(q->oper_name, "") != 0)
      middle_part = Id(SSLstring(q->oper_name));
     else
      middle_part = IdNull;

/*
printf("Make_Vertex_List: parameter_list = %s\n", parameter_list);
*/

     if (strcmp(q->parameter_list, "") != 0)
      suffix_part = make_operator_id_pairs(q->parameter_list);
     else
      suffix_part = OperatorIdPairsNull;

                if (q->met == 0)
                  time = OptionalTimeNull;
                else
                  time = OptionalTime(Time(IntegerVal(C_INT_to_STR0(q->met)),
UnitMS));

                vertex = AVertex(OperatorId(front_part, middle_part, suffix_part),
                                 time);

#ifdef SDE_DEBUG_2
                printf("Make_Vertex_List: operator = %s, id = %d\n", q->name,
(int)vertex);
#endif

                return(VertexListPair(vertex, Make_Vertex_List(p->next)));
            }
         else return( Make_Vertex_List(p->next) );
        )
}
/*-------------------------------------------------------------------------*/
char *get_vertex_type_name(name)
  char *name;
{
  char
    *location_of_dot,
    dummy[100];

  sprintf(dummy, "%s", name);
  location_of_dot = strchr(dummy, '.');
  if (location_of_dot != NULL)
  {
    (*location_of_dot) = '\0';
    return(strdup(dummy));
  }
  else
  {
```

```
    return(strdup(""));
  }
}
/*-------------------------------------------------------------------------*/
char *get_vertex_oper_name(name)
  char *name;
{
  char
    *location_of_dot,
    *location_after_dot,
    *location_of_left_bracket,
    dummy[100];

  sprintf(dummy, "%s", name);
  location_of_dot = strchr(dummy, '.');
  if (location_of_dot != NULL)
  {
    location_after_dot = location_of_dot + 1;
  }
  else
  {
    location_after_dot = dummy;
  }

#ifdef SDE_DEBUG_3
printf("get_vertex_oper_name : string after dot = %s\n", location_after_dot);
#endif

  location_of_left_bracket = strchr(location_after_dot, '(');
  if (location_of_left_bracket != NULL)
  {
    (*location_of_left_bracket) = '\0';
  }

#ifdef SDE_DEBUG_3
printf("get_vertex_oper_name : oper_name length = %d\n", strlen(location_after_dot));
#endif
  return(strdup(location_after_dot));
}

/*-------------------------------------------------------------------------*/
char *get_vertex_parameters(name)
  char *name;
{
  char
    *location_of_left_bracket,
    *location_of_right_bracket,
    *location_of_bar,
    dummy[100];

  sprintf(dummy, "%s", name);

  location_of_left_bracket = strchr(dummy, '(');
  if (location_of_left_bracket == NULL)
  {
    return(strdup(""));
  }
  else
  {
    location_of_bar = strchr(location_of_left_bracket, '|');
    if (location_of_bar == NULL)
    {
      write_error_msg("missing '|' in vertex name %s", name);
                    return(strdup(""));
            }
    else
    {
      location_of_right_bracket = strchr(location_of_bar, ')');
      if (location_of_right_bracket == NULL)
      {
        write_error_msg("missing ')' in vertex name %s", name);
                    return(strdup(""));
      }
      else
      {
        (*location_of_right_bracket) = '\0';
        return(strdup(location_of_left_bracket+1));
      }
    }
  }
}


/*-------------------------------------------------------------------------*/
char *remove_blanks_from_string(name)
  char *name;

{
  char
    clean_string[100],
    dummy[100];

  inti,
    j;

  sprintf(dummy, "%s", name);

  /* remove all blank from the parameter list */

  i = 0;
  j = 0;

  while (dummy[i] != '\0')
  {
    if (dummy[i] != ' ')
    {
      clean_string[j] = dummy[i];
      j++;
    }
    i++;

  }
  clean_string[j] = '\0';

  return(strdup(clean_string));
}

/*-------------------------------------------------------------------------*/
PROD_INSTANCE make_operator_id_pairs(name)
  char *name; /* a non_empty_string containing a vertical bar and 2 id_lists */
{

  char
  *location_of_bar,
  dummy[100];
```

```
  PROD_INSTANCE
  AIDLIST_1,
  AIDLIST_2,
  Extract_AloneIdList_From_String();

  sprintf(dummy, "%s", name);
  location_of_bar = strchr(dummy, '|');
  (*location_of_bar) = '\0';
  AIDLIST_1 = Extract_AloneIdList_From_String(dummy);
  AIDLIST_2 = Extract_AloneIdList_From_String(location_of_bar + 1);
  return(OperatorIdPairs(AIDLIST_1, AIDLIST_2));
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE Extract_AloneIdList_From_String(name)
  char *name;
{
  char
    *i,
    *j,
    dummy[100];

  PROD_INSTANCE
    AID,
    AIDLIST;


  sprintf(dummy, "%s", name);

  /* skip over the delimeters */
  i = dummy;
  while (!( ( (*i) >= 48 && (*i) <= 57 )  /* a decimal digit */
    ||( (*i) >= 65 && (*i) <= 90 )  /* an upper case */
    ||( (*i) >= 97 && (*i) <= 122 )  /* a lower case */
    ||( (*i) == 95 )  /* a '_' char */
    ||( (*i) == '\0' )   /* a null char */
    ) )
  {
    i++;
  }

  if ((*i) == '\0')
    return(AIdNil);
  else
  {
    /* search for end of token */
    j = i;
    while( ( (*j) >= 48 && (*j) <= 57 )  /* a decimal digit */
      ||( (*j) >= 65 && (*j) <= 90 )  /* an upper case */
      ||( (*j) >= 97 && (*j) <= 122 )  /* a lower case */
      ||( (*j) == 95 )  /* a '_' char */
      )
    {
      j++;
    }

    if ((*j) == '\0')
      AIDLIST = AIdNil;
    else
      AIDLIST = Extract_AloneIdList_From_String(j+1);

    (*j) = '\0';
    AID = Id(SSLstring(i));
    return(AIdPair(AID, AIDLIST));
  )
}

/*-------------------------------------------------------------------------*/
/* This function "sweeps" through the STREAM list and genrates the PSDL graph's
   edge list.
*/
/*-------------------------------------------------------------------------*/
PROD_INSTANCE Make_Edge_List(p)
        ST_PTR p;
{
  PROD_INSTANCE
  Get_Operator_Id_Name(),
        v1_id,
  v1_front_part,
  v1_middle_part,
  v1_suffix_part,
        v2_id,
  v2_front_part,
  v2_middle_part,
  v2_suffix_part,
        stream_name,
        stream_time;
  STREAM
        q;
  OPNodePTR
        r1;
  OPERATOR
        s1;
  char
  dummy_name[100],
  *dummy_name_ptr,
  *optional_type_name,
  *oper_name,
  *parameter_list,
  *clean_name,
  *remove_blanks_from_string(),
  *get_vertex_type_name(),
  *get_vertex_oper_name(),
  *get_vertex_parameters();

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Make_Edge_List\n");
#endif

  if (p == NULL)
        return (EdgeListNil);
  else
        (
    q = p->st;  /* q points to the stream record */

    if (strcmp(q->name, "") == 0)
    {  /* need to create dummy name */
    sprintf(dummy_name, "NONAME_%d", Get_Unique_Id());
    q->name = strdup(dummy_name);
    )
    else
    (
    clean_name = remove_blanks_from_string(q->name);
    if (strcmp(clean_name, q->name) != 0)
    {
```

```
      free(q->name);
      q->name = clean_name;
    }
    else
    {
      free(clean_name);
    }
    }

          if (q->is_deleted == FALSE) /* build an edge */
                {

#ifdef SDE_DEBUG_3
printf("Make_Edge_List: edge_name = %s\n", q->name);
#endif

      /* update q's id if q is a new edge */
      if (q->is_new)
      {
        q->id = Get_Unique_Id();
                  q->is_new = FALSE;
                }

      /* Get the first operator's name */

      if (q->from != NULL)
                {
      r1 = q->from; /* r1 points to the edge's first verex head node*/
                s1 = r1->op;  /* s1 points to the "from" vertex's actual record */

#ifdef SDE_DEBUG_3
printf("Make_Edge_List: from vertex name = %s\n", s1->name);
#endif

      clean_name = remove_blanks_from_string(s1->name);
      optional_type_name = get_vertex_type_name(clean_name);
      if (strcmp(optional_type_name, "") != 0)
        v1_front_part = OptionalTypeId(Id(SSLstring(
                    optional_type_name)));
      else
        v1_front_part = OptionalTypeIdNull;


      oper_name = get_vertex_oper_name(clean_name);
      if (strcmp(oper_name, "") != 0)
        v1_middle_part = Id(SSLstring(oper_name));
      else
        v1_middle_part = IdNull;

      parameter_list = get_vertex_parameters(clean_name);
      if (strcmp(parameter_list, "") != 0)
        v1_suffix_part = make_operator_id_pairs(parameter_list);
      else
        v1_suffix_part = OperatorIdPairsNull;

      free(clean_name);
      }
      else
      {
      v1_front_part = OptionalTypeIdNull;
      v1_middle_part = Id(SSLstring("EXTERNAL"));
      v1_suffix_part = OperatorIdPairsNull;
      }

                /* repeat the process for the second vertex */
      if (q->to != NULL)
      {
                r1 = q->to;
                s1 = r1->op;

#ifdef SDE_DEBUG_3
printf("Make_Edge_List: to vertex name = %s\n", s1->name);
#endif

      clean_name = remove_blanks_from_string(s1->name);
      optional_type_name = get_vertex_type_name(clean_name);
      if (strcmp(optional_type_name, "") != 0)
                              v2_front_part = OptionalTypeId(Id(SSLstring(
                                                  optional_type_name)));
                      else
                              v2_front_part = OptionalTypeIdNull;


                      oper_name = get_vertex_oper_name(clean_name);
                      if (strcmp(oper_name, "") != 0)
                              v2_middle_part = Id(SSLstring(oper_name));
                      else
                              v2_middle_part = IdNull;

                      parameter_list = get_vertex_parameters(clean_name);
      if (strcmp(parameter_list, "") != 0)
        v2_suffix_part = make_operator_id_pairs(parameter_list);
      else
        v2_suffix_part = OperatorIdPairsNull;

      free(clean_name);
      }
      else
      {
      v2_front_part = OptionalTypeIdNull;
      v2_middle_part = Id(SSLstring("EXTERNAL"));
      v2_suffix_part = OperatorIdPairsNull;
      }

                  v1_id = OperatorId(v1_front_part, v1_middle_part, v1_suffix_part); /
* first vertex */
                  v2_id = OperatorId(v2_front_part, v2_middle_part, v2_suffix_part); /
* second vertex */

                  stream_name = Id(SSLstring(q->name));         /* PSDL stream name */

                  /* PSDL stream latency time */
      if (q->latency > 0)
                  stream_time = LatencyTime(Time(IntegerVal(C_INT_to_STR0(q->latency)),UnitMS));
      else
      stream_time = LatencyTimeNull;

                  return (EdgeListPair(AnEdge(stream_name,
                                              stream_time,
                                              v1_id,
                                              v2_id),
                                       Make_Edge_List(p->next)));
              }
          else return ( Make_Edge_List(p->next) );
      }
```

```
}
/*-------------------------------------------------------------------------*/
/*                              C-FUNCTIONS                                */
/*-------------------------------------------------------------------------*/
init_selection_action()
{

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering init_selection_action\n");
#endif

}

/*
  Whenever the user clicks on a region of the screen, this function is invoked.
  its primary objective is to determine where in the editing window has the
  user clicked and based upon that, make a decision as to whether or not the
  object clicked upon is displayable.  If it is, we update the external data
  structure; otherwise, we do not do anything.
*/
selection_action(new_selection)
        SELECTION new_selection;
{

  extern VIEW_NO cur_view;
  PROD_INSTANCE
  Change_view();

  void
  Refresh_Graph_Viewer(),
  House_Cleaning();

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering selection_action\n");
#endif


 /*
  if (Global_Set_SdeView == false) {
  Change_view(SSLstring("SDE_VIEW"), SSL_false);
  Global_Set_SdeView = true;
printf("selection_action: Global_Set_SdeView = true");
  }
*/

  House_Cleaning(new_selection);
  Refresh_Graph_Viewer();


#ifdef SDE_DEBUG_1
  printf("Leaving selection_action\n");
#endif
}


/*-------------------------------------------------------------------------*/

void House_Cleaning(new_selection)
        SELECTION new_selection;
{
```

```
extern boolean Global_Enforce_Consistency;

extern PRODUCTION
      prod_prot,
      prod_psdl_pair,
      prod_op,
      prod_data,
      prod_op_spec,
      prod_op_impl,
      prod_t_op_impl,
      prod_input_list,
      prod_inputs,
      prod_output_list,
      prod_outputs,
      prod_type_decl,
      prod_graph,
      prod_a_decl,
      prod_decl,
      prod_stream,
      prod_cc,
      prod_constraints,
      prod_a_constraint;

ATREE
atree = bu_atree(br_buf(cur_browser));

PRODUCTION
top_production,
marker_production;

PROD_INSTANCE
proto,
p,
temp_p,
top_p,
marker_id,
component_id,
op_id,
t_op_impl_id,
Make_IdNull(),
IsTypeDeclNil(),
IsAConstraintNull(),
Get_Id_From_Inputs_List(),
Get_Id_From_Outputs_List(),
Get_Id_From_Stream(),
Get_Operator_Id_From_Constraints(),
Get_Id_From_Type_Decl(),
Find_O_Inputs(),
Find_O_Outputs(),
Find_Stream_Type_Decl(),
Find_A_Constraint(),
Find_TOpImpl_in_operator_impl_list();


HeadPtr
h,
Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(),
Make_Operator_Header();


LINKED_LIST
temp_head,
current_pos_trace = NULL;
```

```
  void
  Enforce_Consistency(),
  Link_To_Structure(),
  Free_Linked_List();

  char
  *name;


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering House_Cleaning\n");
#endif

  if (Global_Enforce_Consistency)
  {
  p = selection_apex(new_selection);

  /* remember the cursor position before fixing the a_tree */
  /* the possible phylum checkpoints are: Prot, Op, Data,  */
  /* InputsListPair, OutputsListPair, Streams, ConstraintsPair */

  temp_p = p;
  current_pos_trace = NULL;

  while (!at_top(temp_p) && (production(temp_p) != prod_op)
    && (production(temp_p) != prod_data)
    && (production(temp_p) != prod_t_op_impl))
  {

#ifdef SDE_DEBUG_3
    /*debugging*/
    printf("House_Cleaning: son_number = %d\n", son_number(temp_p));
#endif

       temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
       temp_head->item_number = son_number(temp_p);
       temp_head->next = current_pos_trace;
       current_pos_trace = temp_head;

       temp_p = father(temp_p);
  }
  top_p = temp_p;

  if (production(top_p) == prod_data)
  {
#ifdef SDE_DEBUG_3
    /*debugging*/
    printf("House_Cleaning: production(top_p) == prod_data\n");
#endif

    top_production = prod_data;
    marker_production = prod_data;
    component_id = id_from_Data(top_p);
  }
  else if (production(top_p) == prod_op)
  {
#ifdef SDE_DEBUG_3
    /*debugging*/
    printf("House_Cleaning: production(top_p) == prod_op\n");
#endif

    top_production = prod_op;
    component_id = id_from_Op(top_p);

    if (IntValue(IdIsNull(component_id)) == 0)
    {
      h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(top_p);
      if (h == NULL)
      {

#ifdef SDE_DEBUG_3
        /*debugging*/
        printf("House_Cleaning: cannot find header_node\n");
#endif

        /* need to make new header node */
        h = Make_Operator_Header(str0_to_str_ro(StrValue(Get_Id(id_from_Op(top_p)))),
                  NULL, NULL, Get_Unique_Id(), (int)top_p, FALSE);
        Link_To_Structure(h);
      }

      if (!(h->multi_op_error))
      {
        temp_p = p;
        while ((temp_p != top_p)
            && (production(temp_p) != prod_input_list)
            && (production(temp_p) != prod_output_list)
            && (production(temp_p) != prod_stream)
            && (production(temp_p) != prod_constraints))
        {
            temp_p = father(temp_p);
        }


        if ((production(temp_p) == prod_input_list) &&  h->input_error)
        {
#ifdef SDE_DEBUG_3
            /*debugging*/
            printf("House_Cleaning: production(temp_p) == prod_input_list\n");
#endif

            Free_Linked_List(current_pos_trace);
            marker_production = prod_input_list;
            marker_id = Get_Id_From_Inputs_List(p);
        }
        else if ((production(temp_p) == prod_output_list) && h->output_error)
        {
#ifdef SDE_DEBUG_3
            /*debugging*/
            printf("House_Cleaning: production(temp_p) == prod_output_list\n");
#endif

            Free_Linked_List(current_pos_trace);
            marker_production = prod_output_list;
            marker_id = Get_Id_From_Outputs_List(p);
        }
        else if ((production(temp_p) == prod_stream) && h->stream_error)
        {
#ifdef SDE_DEBUG_3
            /*debugging*/
            printf("House_Cleaning: production(temp_p) == prod_stream\n");
#endif

            Free_Linked_List(current_pos_trace);
```

```
#ifdef SDE_DEBUG_3
            printf("House_Cleaning: pass Free_Linked_List\n");
#endif
            marker_production = prod_stream;

            temp_p = p;
            while (!at_top(temp_p) && (production(temp_p) != prod_type_decl))
            {
                temp_p = father(temp_p);
            }
#ifdef SDE_DEBUG_3
            printf("House_Cleaning: pass while loop\n");
#endif
            if (production(temp_p) == prod_type_decl)
            {
                marker_id = Get_Id_From_Type_Decl(temp_p);
#ifdef SDE_DEBUG_3
                printf("House_Cleaning: marker_id = %s\n",
                    str0_to_str_ro(StrValue(Get_Id(marker_id))));
#endif
            }
            else
            {
                marker_id = Make_IdNull();
#ifdef SDE_DEBUG_3
                printf("House_Cleaning: marker_id = IdNull\n");
#endif
            }
#ifdef SDE_DEBUG_3
            printf("House_Cleaning: pass Get_Id_From_Type_Decl\n");
#endif
        }
        else if ((production(temp_p) == prod_constraints) && h->constraint_error)
        {
#ifdef SDE_DEBUG_3
            /*debugging*/
            printf("House_Cleaning: production(temp_p) == prod_constraints\n");
#endif

            Free_Linked_List(current_pos_trace);
               marker_production = prod_constraints;
            marker_id = Get_Operator_Id_From_Constraints(p);
        }
        else
            marker_production = top_production;

      }
    }
  }
  else if (production(top_p) == prod_t_op_impl)
  {
#ifdef SDE_DEBUG_3
    printf("House_Cleaning: (production(top_p) == prod_t_op_impl)\n");
#endif

    top_production = prod_t_op_impl;

    t_op_impl_id = id_from_TOpImpl(top_p);

#ifdef SDE_DEBUG_3
    printf("House_Cleaning: t_op_impl_id = %s\n",
str0_to_str_ro(StrValue(Get_Id(t_op_impl_id))));
#endif

    /* get component id */
    temp_p = top_p;
    while (production(temp_p) != prod_data)
    {
      temp_p = father(temp_p);
    }
    component_id = id_from_Data(temp_p);

#ifdef SDE_DEBUG_3
      printf("House_Cleaning: component_id = %s\n",
str0_to_str_ro(StrValue(Get_Id(component_id))));
#endif

    /* get header node */
    if (IntValue(IdIsNull(t_op_impl_id)) == 0)
    {
      h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(top_p);
                    if (h == NULL)
                    {

#ifdef SDE_DEBUG_3
                        /*debugging*/
                        printf("House_Cleaning: cannot find header_node\n");
#endif

                        /* need to make new header node */
                        h =
Make_Operator_Header(str0_to_str_ro(StrValue(Get_Id(id_from_TOpImpl(top_p)))),
                                                    NULL, NULL, Get_Unique_Id(),
(int)top_p, FALSE);
                        Link_To_Structure(h);
                    }


      /* see if cursor is positioned at streams or constraints */
                    temp_p = p;
                    while ((temp_p != top_p)
                        && (production(temp_p) != prod_stream)
                        && (production(temp_p) != prod_constraints))
                    {
                        temp_p = father(temp_p);
                    }


                    if ((production(temp_p) == prod_stream) && (h->stream_error))
                    {
#ifdef SDE_DEBUG_3
                        /*debugging*/
        printf("House_Cleaning: production(temp_p) == prod_stream\n");
#endif
                        Free_Linked_List(current_pos_trace);

#ifdef SDE_DEBUG_3
                        printf("House_Cleaning: pass Free_Linked_List\n");
#endif
                        marker_production = prod_stream;

                        temp_p = p;
                        while ((temp_p != top_p) && (production(temp_p) !=
prod_type_decl))
                        {
                                temp_p = father(temp_p);
```

```
                                )
#ifdef SDE_DEBUG_3
                                printf("House_Cleaning: pass while loop\n");
#endif
                                if (production(temp_p) == prod_type_decl)
                                {
                                        marker_id = Get_Id_From_Type_Decl(temp_p);
#ifdef SDE_DEBUG_3
                                        printf("House_Cleaning: marker_id = %s\n",

str0_to_str_ro(StrValue(Get_Id(marker_id))));
#endif
                                }
                                else
                                {
                                        marker_id = Make_IdNull();
#ifdef SDE_DEBUG_3
                                        printf("House_Cleaning: marker_id =

IdNull\n");
#endif
                                }
#ifdef SDE_DEBUG_3
                                printf("House_Cleaning: pass Get_Id_From_Type_Decl\n");
#endif
                        }
                        else if ((production(temp_p) == prod_constraints) && (h-
>constraint_error))
                        {
                                /*debugging*/
printf("House_Cleaning: production(temp_p) == prod_constraints\n");

                                Free_Linked_List(current_pos_trace);
                                marker_production = prod_constraints;
                                marker_id = Get_Operator_Id_From_Constraints(p);
                        }
                        else
                                marker_production = top_production;

                }
    else
    {
      marker_production = top_production;
      Free_Linked_List(current_pos_trace);
    }
        }
  else
  {
    component_id = Make_IdNull();
  }

  Enforce_Consistency();

#ifdef SDE_DEBUG_3
printf("House_Cleaning: After Enforce_Consistency, component_id = %s\n",
  str0_to_str_ro(StrValue(Get_Id(component_id))));
#endif

  if (IntValue(IdIsNull(component_id)) == 0)
  {
    /* reset curson position */
    if ((top_production == prod_data)
      || (top_production == prod_op)
      || (top_production == prod_t_op_impl))
    {
      top_p = Find_Component(component_id, psdl_components_from_Prot(Global_Proto));


#ifdef SDE_DEBUG_3
      /*debugging*/
      printf("House_Cleaning: component_id = %s\n",
            str0_to_str_ro(StrValue(Get_Id(component_id))));
#endif

      if ((top_production == prod_data) ||
        ((top_production == prod_op) && !(h->multi_op_error)))
      {

#ifdef SDE_DEBUG_3
        /*debugging*/
        printf("House_Cleaning: top_p_id = %s\n",
            str0_to_str_ro(StrValue(Get_Id(id_from_Op_or_Data(top_p)))));
/*
            str0_to_str_ro(StrValue(Get_Id(son(top_p, 1)))));
*/
#endif
        temp_p = top_p;

        if (marker_production == top_production)
        {
#ifdef SDE_DEBUG_3
            /*debugging*/
            printf("House_Cleaning: marker_production == top_production\n");
#endif

            temp_head = current_pos_trace;

            while (temp_head != NULL)
            {
                temp_p = son(temp_p, (temp_head->item_number));
                temp_head = temp_head->next;
            }
            p = temp_p;
            Free_Linked_List(current_pos_trace);
        }
        else if (marker_production == prod_input_list)
        {
#ifdef SDE_DEBUG_3
            /*debugging*/
            printf("House_Cleaning: marker_production == prod_input_list\n");
#endif

/*
            temp_p = son(son(temp_p, 2), 2);
*/
            temp_p = o_inputs_list_from_OperatorSpec(
                operator_spec_from_Op(temp_p));
            if (IntValue(IdIsNull(marker_id)) == 0)
            {
#ifdef SDE_DEBUG_3
                /*debugging*/
                printf("House_Cleaning: marker_id = %s\n",
                    str0_to_str_ro(StrValue(Get_Id(marker_id))));
#endif

                p = Find_O_Inputs(marker_id, temp_p);
                if (BoolValue(IsTypeDeclNil(p)))
```

```
                {
                    p = temp_p;
                }
            }
            else
            {
                p = temp_p;
            }
        }
        else if (marker_production == prod_output_list)
        {
#ifdef SDE_DEBUG_3
            /*debugging*/
            printf("House_Cleaning: marker_production == prod_output_list\n");
#endif

/*
            temp_p = son(son(temp_p, 2), 3);
*/
            temp_p = o_outputs_list_from_OperatorSpec(
                    operator_spec_from_Op(temp_p));
            if (IntValue(IdIsNull(marker_id)) == 0)
            {
#ifdef SDE_DEBUG_3
                /*debugging*/
                printf("House_Cleaning: marker_id = %s\n",
                    str0_to_str_ro(StrValue(Get_Id(marker_id))));
#endif

                p = Find_O_Outputs(marker_id, temp_p);

#ifdef SDE_DEBUG_3
                /*debugging*/
                printf("House_Cleaning: returning from Find_O_Outputs\n");
#endif

                if (BoolValue(IsTypeDeclNil(p)))
                {
                    p = temp_p;
                }
            }
            else
            {
                p = temp_p;
            }
        }
        else if (marker_production == prod_stream)
        {
#ifdef SDE_DEBUG_3
            /*debugging*/
            printf("House_Cleaning: marker_production == prod_stream\n");
#endif

/*
            temp_p = son(son(son(temp_p, 3), 2), 1);*/
            temp_p = optional_streams_from_Declarations(
                    declarations_from_OperatorImpl(
                      operator_impl_from_Op(temp_p)));

            if (IntValue(IdIsNull(marker_id)) == 0)
            {
#ifdef SDE_DEBUG_3
                /*debugging*/
                printf("House_Cleaning: marker_id = %s\n",
                    str0_to_str_ro(StrValue(Get_Id(marker_id))));
#endif
                p = Find_Stream_Type_Decl(marker_id, temp_p);
                if (BoolValue(IsTypeDeclNil(p)))
                {
                    p = temp_p;
                }
            }
            else
            {
                p = temp_p;
            }
        }
        else if (marker_production == prod_constraints)
        {
#ifdef SDE_DEBUG_3
            /*debugging*/
            printf("House_Cleaning: marker_production == prod_constraints\n");
#endif

            temp_p = constraints_from_Cc(
                    cc_from_OperatorImpl(
                      operator_impl_from_Op(temp_p)));
            if (IntValue(OperatorIdIsNull(marker_id)) == 0)
            {
#ifdef SDE_DEBUG_3
                /*debugging*/
                printf("House_Cleaning: marker_id = %s\n",
                    str0_to_str_ro(StrValue(Get_Id(id_from_OperatorId(marker_id)))));
#endif

                p = Find_A_Constraint(marker_id, temp_p);
                if (BoolValue(IsAConstraintNull(p)))
                {
                    p = temp_p;
                }
            }
            else
            {
                p = temp_p;
            }
        }
      }
      else if (top_production == prod_t_op_impl)
      {
#ifdef SDE_DEBUG_3
printf("House_Cleaing: after Enforce_Consistency\n");
printf("House_Cleaning: top_production == prod_t_op_impl\n");
printf("House_Cleaning: t_op_impl_id = %s\n",
  str0_to_str_ro(StrValue(Get_Id(t_op_impl_id))));
#endif

        if (IntValue(IdIsNull(t_op_impl_id))== 0)
        {
            temp_p = Find_TOpImpl_in_operator_impl_list(
                    t_op_impl_id,
                    operator_impl_list_from_TypeImpl(
                      type_impl_from_Data(top_p)));

            if (production(temp_p) == prod_t_op_impl)
            {
#ifdef SDE_DEBUG_3
```

```
printf("House_Cleaning: after Find_TOpImpl_in_operator_impl_list, t_op_impl_id = %s\n",
  str0_to_str_ro(StrValue(Get_Id(id_from_TOpImpl(temp_p)))));
#endif

                if (marker_production == top_production)
                        {
#ifdef SDE_DEBUG_3
                                /*debugging*/
                    printf("House_Cleaning: marker_production == top_production\n");
#endif

                                temp_head = current_pos_trace;

                                while (temp_head != NULL)
                                {
                                        temp_p = nth(temp_p,temp_head->item_number);
                                        temp_head = temp_head->next;
                                }
                                p = temp_p;
                                Free_Linked_List(current_pos_trace);
                        }
                else if (marker_production == prod_stream)
                {
#ifdef SDE_DEBUG_3
                    printf("House_Cleaning: marker_production == prod_stream\n");
#endif
                    temp_p = optional_streams_from_Declarations(
                                        declarations_from_OperatorImpl(
                                            operator_impl_from_TOpImpl(temp_p)));

                                if (IntValue(IdIsNull(marker_id)) == 0)
                                {
                                        /*debugging*/
#ifdef SDE_DEBUG_3
                                printf("House_Cleaning: marker_id = %s\n",

str0_to_str_ro(StrValue(Get_Id(marker_id))));
#endif
                                        p = Find_Stream_Type_Decl(marker_id,temp_p);
                                        if (BoolValue(IsTypeDeclNil(p)))
                                        {
                                                p = temp_p;
                                        }
                                }
                                else
                                {
                                        p = temp_p;
                                }
                        }
                        else if (marker_production == prod_constraints)
                        {
                                /*debugging*/
#ifdef SDE_DEBUG_3
                    printf("House_Cleaning: marker_production == prod_constraints\n");
#endif

                                temp_p = constraints_from_Cc(
                                        cc_from_OperatorImpl(
                                            operator_impl_from_TOpImpl(temp_p)));
                                if (IntValue(IdIsNull(marker_id)) == 0)
                                {
                                        /*debugging*/
#ifdef SDE_DEBUG_3
                                        printf("House_Cleaning: marker_id = %s\n",

str0_to_str_ro(StrValue(Get_Id(id_from_OperatorId(marker_id)))));
#endif
                                        p = Find_A_Constraint(marker_id, temp_p);
                                        if (BoolValue(IsAConstraintNull(p)))
                                        {
                                                p = temp_p;
                                        }
                                }
                                else
                                {
                                        p = temp_p;
                                }
                        }
                    }
                    else
                    {
                        p = type_impl_from_Data(top_p);
                    }
                }
                else
                {
                    p = type_impl_from_Data(top_p);
                }
            }
            else
            {
#ifdef SDE_DEBUG_3
                /*debugging*/
                printf("House_Cleaning: multi-op-error in %s\n",
                    str0_to_str_ro(StrValue(Get_Id(id_from_Op(top_p)))));
#endif
                temp_p = top_p;
                p = Global_Proto;
                Free_Linked_List(current_pos_trace);
            }
        }
        else
        {
          p = Global_Proto;
          Free_Linked_List(current_pos_trace);
        }
      }



       /* update atree, buffer and selection */
       temp_p = one_point_selection(p);
       move_selection(atree, temp_p);
       br_set_insert_pt_to_selection(cur_browser);
       cmd_cond_modifies(cur_browser, cur_buffer);

       br_paint_all();

      }
}

/*-------------------------------------------------------------------------*/
void Refresh_Graph_Viewer()
{
  HeadPtr
```

```
  h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(),
  Make_Operator_Header();

   PROD_INSTANCE
  IsIdNull(),
  component_id,
  t_op_impl_id,
  op_p,
  t_op_impl_p,
  temp_p;

   void
  Update_Operator(),
  Move_To_Structure_Front();

#ifdef SDE_DEBUG_1
  printf("Entering Refresh_Graph_Viewer\n");
#endif


  temp_p = selection_apex(SE_selection(bu_atree(br_buf(cur_browser))));
  while (!at_top(temp_p) && (production(temp_p) != prod_op) && (production(temp_p) !=
prod_t_op_impl))
  {
    temp_p = father(temp_p);
  }

  if (!at_top(temp_p))
  {
    if (production(temp_p) == prod_op)
    {

      op_p = temp_p;
      component_id = id_from_Op(op_p);

      if (IntValue(IdIsNull(component_id)) == 0)
                {
                        h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(op_p);
                        if (h == NULL)
                        {

#ifdef SDE_DEBUG_3
                                /*debugging*/
                                printf("Refresh_Graph_Viewer: cannot find header_node\n");
#endif

                                /* need to make new header node */
                Make_Operator_Header(str0_to_str_ro(StrValue(Get_Id(id_from_Op(op_p)))),
                                                        NULL, Get_Unique_Id((int)op_p), FALSE);
                                Link_To_Structure(h);
                        }

        /* to order the header nodes so that most-recently-used-first */
        Move_To_Structure_Front(h);


        /* see if need to refresh graph view */
        if (strcmp(str0_to_str_ro(StrValue(Get_Id(component_id))),
                                Global_Current_Op_Name) != 0)
        {
            /* cursor is at a different operator */
            if (strcmp(Global_Current_Op_Name, "") != 0)
            {
                free(Global_Current_Op_Name);
            }
            Global_Current_Op_Name = strdup(str0_to_str_ro(
                                StrValue(Get_Id(
                                id_from_Op(op_p)))));
/*
                                StrValue(Get_Id(son(op_p, 1)))));
*/
            Global_Refresh_Graph_Viewer = true;
        }
        else
        {
            temp_p = selection_apex(SE_selection(bu_atree(br_buf(cur_browser))));
            while (!at_top(temp_p) && (production(temp_p) != prod_graph)
                                && (production(temp_p) != prod_graph_null))
            {
                temp_p = father(temp_p);
            }

            if ((production(temp_p) == prod_graph) ||
                    (production(temp_p) == prod_graph_null))
            {
                /* cursor is at the same operator but graph is selected*/
                Global_Refresh_Graph_Viewer = true;
            }

        }
      }
    }
    else
    {
      op_p = temp_p;

      t_op_impl_p = temp_p;
      t_op_impl_id = id_from_TOpImpl(t_op_impl_p);

      if (IntValue(IdIsNull(t_op_impl_id)) == 0)
                {
                        h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(t_op_impl_p);
                        if (h == NULL)
                        {

#ifdef SDE_DEBUG_3
                                /*debugging*/
                                printf("Refresh_Graph_Viewer: cannot find header_node\n");
#endif

                                /* need to make new header node */
                                h =
Make_Operator_Header(str0_to_str_ro(StrValue(Get_Id(id_from_TOpImpl(t_op_impl_p)))),
                                                        NULL, Get_Unique_Id((int)op_p), FALSE);
                                Link_To_Structure(h);
                        }

        /* to order the header nodes so that most-recently-used-first */
        Move_To_Structure_Front(h);


        /* see if need to refresh graph view */
        if (strcmp(str0_to_str_ro(StrValue(Get_Id(t_op_impl_id))),
                                Global_Current_Op_Name) != 0)
        {
```

```
            /* cursor is at a different operator */
            if (strcmp(Global_Current_Op_Name, "") != 0)
            {
                free(Global_Current_Op_Name);
            }
            Global_Current_Op_Name = strdup(str0_to_str_ro(
                                StrValue(Get_Id(
                                t_op_impl_id))));
            Global_Refresh_Graph_Viewer = true;
        }
        else
        {
            temp_p = selection_apex(SE_selection(bu_atree(br_buf(cur_browser))));
            while (!at_top(temp_p) && (production(temp_p) != prod_graph)
                                && (production(temp_p) != prod_graph_null))
            {
                temp_p = father(temp_p);
            }

            if ((production(temp_p) == prod_graph) ||
                    (production(temp_p) == prod_graph_null))
            {
                /* cursor is at the same operator but graph is selected*/
                Global_Refresh_Graph_Viewer = true;
            }

        }
      }
    }


    if (Global_Refresh_Graph_Viewer)
          {

      Global_Refresh_Graph_Viewer = false;

          Update_Operator(op_p, h);
          current_graph = h;
      the_operator_list = h->operator_list;
      the_stream_list = h->stream_list;

#ifdef SDE_DEBUG_3
                printf("--OPERATOR FOUND IN THE LIST--\n");
                printf("%s\n", h->name);
                Print_OperatorS_Operators(h->operator_list);
                Print_OperatorS_Streams(h->stream_list);
#endif

           refresh();
    }
  }


#ifdef SDE_DEBUG_1
  printf("Leaving Refresh_Graph_Viewer\n");
#endif
}

/*-------------------------------------------------------------------------*/
void Update_Operator(p, q)
  PROD_INSTANCE
  p;
  HeadPtr
  q;
{
  int
  x,  /* for the x coordinate of the new vertices when needed */
  y;  /* for the y coordinate of the new vertices when needed */

#ifdef SDE_DEBUG_1
  printf("Entering Update_Operator\n");
#endif


  x = 80;
  y = 45;

  /* update operator list */
  Update_Operator_List(p, q, &x, &y);


#ifdef GRAPHICS_DEBUG
  printf("finishing Update_Operator_List\n");
  Print_OperatorS_Operators(q->operator_list);
  Print_OperatorS_Streams(q->stream_list);
#endif

  x = 35;
  y = 30;

  /* update stream list */
  Update_Stream_List(p, q, &x, &y);


#ifdef GRAPHICS_DEBUG
  printf("finishing Update_Stream_List\n");
                Print_OperatorS_Operators(q->operator_list);
                Print_OperatorS_Streams(q->stream_list);
#endif

}
/*-------------------------------------------------------------------------*/
Update_Operator_List(p, q, x_ptr, y_ptr)
  PROD_INSTANCE
  p;
  HeadPtr
  q;
  int
  *x_ptr, /* for the x coordinate of the new vertices when needed */
  *y_ptr; /* for the y coordinate of the new vertices when needed */

{
  PROD_INSTANCE
    vertex_list,
    vertex,
  Get_Operator_Impl(),
  Get_Vertex_List(),
  Get_Vertex(),
  Rest_Of_Vertex_List();

  OPNodePTR
  v;


#ifdef SDE_DEBUG_1
  printf("Entering Update_Operator_List\n");
```

```
#endif;

  /* initialize the "is_deleted" and "is_modified" fields in all operato's
     in q->operator_list */
  v = q->operator_list;
  while (v != NULL)
  {
  v->op->is_deleted = TRUE;
  v->op->is_modified = FALSE;
        v = v->next;
  }

  if (production(p) == prod_op)
  vertex_list = Get_Vertex_List(operator_impl_from_Op(p));
  else
  vertex_list = Get_Vertex_List(operator_impl_from_TOpImpl(p));


  while (production(vertex_list) == prod_vertex_list_pair)
  {
    vertex = Get_Vertex(vertex_list);
    vertex_list = Rest_Of_Vertex_List(vertex_list);
    Update_Vertex(vertex, q, x_ptr, y_ptr);
  }
}
/*-------------------------------------------------------------------------*/
Update_Vertex(v, h, x_ptr, y_ptr)
  PROD_INSTANCE
    v;
  HeadPtr
    h;
  int
  *x_ptr,  /* the x coordinate for the new operator if one is needed */
  *y_ptr;  /* the y coordinate for the new operator if one is needed */
{
  OPNodePTR
  q,
  Make_Operator_Node(),
  new_vertex,
  k,
  Vtx_Name_Is_In_List();

  HeadPtr
  f,
  Make_Operator_Header(),
  Op_Name_Is_In_List();

  BOOL
  is_composite;

  char
  dummy[100],
  *vertex_type_name,
  *vertex_oper_name,
  *Get_Parameters_From_Operator_Id_Pairs(),
  *vertex_parameters;

  int
  Get_Met(),
  met;

  void
  Link_Operator(),
  Link_To_Structure();

  PROD_INSTANCE
  vertex_operator_id_pairs,
  Get_Vertex_Type_Id_Name(),
  Get_Vertex_Operator_Id_Name(),
  Get_Vertex_OpIdPairs();


#ifdef SDE_DEBUG_1
  printf("Entering Update_Vertex\n");
#endif;

  vertex_type_name = str0_to_str_ro(StrValue(Get_Vertex_Type_Id_Name(v)));
  vertex_oper_name = str0_to_str_ro(StrValue(Get_Vertex_Operator_Id_Name(v)));
  vertex_operator_id_pairs = Get_Vertex_OpIdPairs(v);
  vertex_parameters = Get_Parameters_From_Operator_Id_Pairs(
                      vertex_operator_id_pairs);

  if (strcmp(vertex_type_name, "") == 0)
  sprintf(dummy, "%s", vertex_oper_name);
  else
  {
  sprintf(dummy, "%s.%s(%s)", vertex_type_name, vertex_oper_name,
                                    vertex_parameters);
  }


  /* get "is_composite" value from the corresponding
     op_node if one exists. */
  f = Op_Name_Is_In_List(vertex_oper_name);
  if (f == NULL)
  {
        /* need to create a new header for the vertex */
  f = Make_Operator_Header(vertex_oper_name, NULL, NULL, Get_Unique_Id(),
            fake_prod_no, fake_is_deleted);

  if (strcmp(vertex_type_name, "") != 0)
    f->type_id = strdup(vertex_type_name);

  Link_To_Structure(f);
  }

  if (strcmp(vertex_type_name, "") == 0)
  {
  is_composite = f->is_composite;
  }
  else
  {/* an instance of a type-operator */
  is_composite = false;
  }

  met = Get_Met(v);

  /* see if vertex is already in the operator_list */
  k = Vtx_Name_Is_In_List(vertex_type_name, vertex_oper_name, vertex_parameters, h-
>operator_list);

  if (k != NULL)
  {
  k->op->met = met;
  k->op->is_composite = is_composite;
  k->op->is_deleted = FALSE;
```

```
  }
  else
  {
   /* new vertex to be inserted */

#ifdef GRAPHICS_DEBUG
     /* debugging */
     printf("Update_Vertex: create new vertex (%s)\n", name);
#endif

     new_vertex = Make_Operator_Node(dummy,
            vertex_type_name,
            vertex_oper_name,
            vertex_parameters,
                                    Get_Unique_Id(),
                                    met,
                                    *x_ptr,
                                    *y_ptr,
                                    fake_radius,
                                    fake_color,
                                    fake_font, /* name_font */
                                    (*x_ptr + 10), /* name_x */
                                    (*y_ptr + 34), /* name_y */
                                    fake_font, /* met_font */
                                    *x_ptr, /* met_x */
                                    *y_ptr, /* met_y */
                                    fake_is_deleted,
                                    fake_is_new,
                                    is_composite,
                                    fake_is_terminator,
                                    fake_is_modified);
      Link_Operator(new_vertex, h);

      if (*x_ptr < 600)
      {
    *x_ptr = *x_ptr + 80;
    if (((*x_ptr / 80) % 2) == 0)
    *y_ptr = *y_ptr + 50;
    else
    *y_ptr = *y_ptr - 40;
      }
      else
      {
    *x_ptr = 80;
    *y_ptr = *y_ptr + 80;
      }
  }
}
/*-------------------------------------------------------------------------------*/
Update_Stream_List(p, q, x_ptr, y_ptr)
  PROD_INSTANCE
  p;
  HeadPtr
  q;
  int
  *x_ptr,  /* the x coordinate for the new operator if one is needed */
  *y_ptr;  /* the y coordinate for the new operator if one is needed */
{
  PROD_INSTANCE
  temp_p,
  Get_Id(),
  FirstElement(),
  IdSetTail(),
    edge_list,
    edge,
  Get_Operator_Spec(),
  Get_Operator_Impl(),
  Get_State_List(),
  Get_Edge_List(),
  Get_Edge(),
  Rest_Of_Edge_List();

  PRODUCTION
  edge_list_nil;

  ST_PTR
  v;

  /* initialize the "is_deleted" and "is_modified" fields of all edges in
     q->stream_list */
  v = q->stream_list;
  while (v != NULL)
  {
  v->st->is_deleted = TRUE;
  v->st->is_modified = FALSE;
  v = v->next;
  }


  edge_list_nil = op_search("EdgeListNil");
  if (production(p) == prod_op)
  edge_list = Get_Edge_List(operator_impl_from_Op(p));
  else
  edge_list = Get_Edge_List(operator_impl_from_TOpImpl(p));

#ifdef SDE_DEBUG_3
temp_p = q->state_id_set;
while (!BoolValue(IsNull(temp_p)))
{
  printf("Update_Edge_List: state_id = %s\n",
    str0_to_str_ro(StrValue(Get_Id(FirstElement(temp_p)))));
  temp_p = IdSetTail(temp_p);
}
#endif

  while (production(edge_list) != edge_list_nil)
  {
    edge = Get_Edge(edge_list);
    edge_list = Rest_Of_Edge_List(edge_list);
    Update_Edge(edge, q, x_ptr, y_ptr);
  }
}
/*-------------------------------------------------------------------------------*/
Update_Edge(e, h, x_ptr, y_ptr)
  PROD_INSTANCE
  e;
  HeadPtr
   h;
  int
  *x_ptr,  /* the x coordinate for external label if one is needed */
  *y_ptr;  /* the y coordinate for external label if one is needed */
{
  int
  latency,
  Get_Latency();
```

```
  char
  *Get_Parameters_From_Operator_Id_Pairs(),
  *optional_type_name,
  *oper_name,
  *parameter_list,
  *edge_name;

  BOOL
  is_state;

  ST_PTR
  Make_Stream_Node(),
  new_edge;

  STREAM
  Edge_Name_Is_In_List(),
  k;

  SPLINE_PTR
  spl_ptr;

  OPNodePTR
  Vtx_Name_Is_In_List(),
  from,
  to,
  op_list;

  void
  Link_Stream();

  int
        from_x,
        from_y,
        to_x,
        to_y;

  PROD_INSTANCE
  temp_p,
  FirstElement(),
  IdSetTail(),
  Get_Id(),
  Get_Edge_From_Vertex_Type_Id_Name(),
  Get_Edge_From_Vertex_Operator_Id_Name(),
  Get_Edge_From_Vertex_OpIdPairs(),
  Get_Edge_To_Vertex_Type_Id_Name(),
  Get_Edge_To_Vertex_Operator_Id_Name(),
  Get_Edge_To_Vertex_OpIdPairs(),
  from_vertex_id,
  to_vertex_id,
  IsElement();

#ifdef SDE_DEBUG_1
  printf("Entering Update_Edge\n");
#endif

  edge_name = strdup(str0_to_str_ro(StrValue(Get_Edge_Name(e))));
  from_vertex_id = Get_Edge_From_Vertex_Op_Id(e);
  to_vertex_id = Get_Edge_To_Vertex_Op_Id(e);

#ifdef SDE_DEBUG_3
printf("Update_Edge: edge name = %s\n", edge_name);
#endif

  optional_type_name = str0_to_str_ro(StrValue(
        Get_Edge_From_Vertex_Type_Id_Name(from_vertex_id)));

#ifdef SDE_DEBUG_3
printf("Update_Edge: from vertex optional_type_name = %s\n", optional_type_name);
#endif

  oper_name = str0_to_str_ro(StrValue(
        Get_Edge_From_Vertex_Operator_Id_Name(from_vertex_id)));

#ifdef SDE_DEBUG_3
printf("Update_Edge: from vertex oper_name = %s\n", oper_name);
#endif

  parameter_list = Get_Parameters_From_Operator_Id_Pairs(
        Get_Edge_From_Vertex_OpIdPairs(from_vertex_id));

#ifdef SDE_DEBUG_3
printf("Update_Edge: from vertex parameter_list = %s\n", parameter_list);
#endif

  if (strcmp(oper_name, "EXTERNAL") != 0)
  from = Vtx_Name_Is_In_List(optional_type_name, oper_name,
                parameter_list, h->operator_list);
  else
  from = NULL;

  if (from == NULL)
  {
  /* from EXTERNAL */
  from_x = *x_ptr;
  from_y = *y_ptr;

  *y_ptr = *y_ptr + 40;
  }
  else
  {
  from_x = from->op->x;
  from_y = from->op->y;
  }


  optional_type_name = str0_to_str_ro(StrValue(
        Get_Edge_To_Vertex_Type_Id_Name(to_vertex_id)));

#ifdef SDE_DEBUG_3
printf("Update_Edge: to vertex optional_type_name = %s\n", optional_type_name);
#endif

  oper_name = str0_to_str_ro(StrValue(
        Get_Edge_To_Vertex_Operator_Id_Name(to_vertex_id)));

#ifdef SDE_DEBUG_3
printf("Update_Edge: to vertex oper_name = %s\n", oper_name);
#endif

  parameter_list = Get_Parameters_From_Operator_Id_Pairs(
        Get_Edge_To_Vertex_OpIdPairs(to_vertex_id));

#ifdef SDE_DEBUG_3
printf("Update_Edge: to vertex parameter_list = %s\n", parameter_list);
#endif
```

```
  if (strcmp(oper_name, "EXTERNAL") != 0)
  to = Vtx_Name_Is_In_List(optional_type_name, oper_name,
                parameter_list, h->operator_list);
  else
  to = NULL;

  if (to == NULL)
  {
  /* to EXTERNAL */
  to_x = *x_ptr;
  to_y = *y_ptr;

  *y_ptr = *y_ptr + 40;
  }
  else
  {
  to_x = to->op->x;
  to_y = to->op->y;
  }



  latency = Get_Latency(e);

#ifdef SDE_DEBUG_3
temp_p = h->state_id_set;
while (!BoolValue(IsNull(temp_p)))
{
  printf("Update_Edge: state_id = %s\n",
    str0_to_str_ro(StrValue(Get_Id(FirstElement(temp_p)))));
  temp_p = IdSetTail(temp_p);
}
#endif

  is_state = BoolValue(IsElement(id_from_AnEdge(e), h->state_id_set));

#ifdef SDE_DEBUG_3
printf("Update_Edge: %s is_state = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_AnEdge(e)))),
    (is_state)?"TRUE":"FALSE");

temp_p = h->state_id_set;
while (!BoolValue(IsNull(temp_p)))
{
  printf("Update_Edge: state_id = %s\n",
    str0_to_str_ro(StrValue(Get_Id(FirstElement(temp_p)))));
  temp_p = IdSetTail(temp_p);
}
#endif

  k = Edge_Name_Is_In_List(edge_name, from, to, h->stream_list);

  if (k != NULL)
  {
  k->latency = latency;
  k->is_state_variable = is_state;
/*
        k->from = from;
        k->to = to;
*/
  k->is_deleted = FALSE;
  }
  else
  {

#ifdef GRAPHICS_DEBUG
  /* debugging */
  printf("edge_name = %s\n", edge_name);
  printf("from_x = %d, to_x = %d\n", from_x, to_x);
#endif

  if ((from == NULL) || (to == NULL))
  {
    /* External streams must have at least one control point */
    spl_ptr = (SPLINE_PTR) malloc(sizeof(SPLINE_NODE));
    spl_ptr->next = NULL;
    if (from ==  NULL)
    {
      spl_ptr->x = from_x;
      spl_ptr->y = from_y;
    }
    else
    {
      spl_ptr->x = to_x;
      spl_ptr->y = to_y;
    }
  }
  else
  {
    spl_ptr = NULL;
  }

  /* need to create new stream node */
  new_edge = Make_Stream_Node(edge_name,
            Get_Unique_Id(),
            from,
            to,
                                        fake_font, /* name_font */
            ((int)(from_x + to_x)/2),  /* name_x */
            ((int)(from_y + to_y)/2 + 30),  /* name_y */
                                        fake_font, /* latency_font */
            ((int)(from_x + to_x)/2),  /* latency_x */

            ((int)(from_y + to_y)/2 + 45),  /* latency_y */
            spl_ptr,
            latency,
            fake_is_deleted,
            fake_is_new,
            fake_is_modified,
            is_state);
  Link_Stream(new_edge, h);
  }
}
/*-------------------------------------------------------------------------*/
void Add_Input_Output_State_Nodes(h)
  HeadPtr h;
{
  int
  input_count,
  output_count,
  state_count,
  first_x,
  node_x,
  edge_label_y,
  edge_x;
```

```
OPNodePTR
Vtx_Name_Is_In_List(),
Make_Operator_Node(),
new_vertex,
k;

STREAM
Edge_Name_Is_In_List(),
t;

ST_PTR
new_edge,
Make_Stream_Node();

SPLINE_PTR
spl_ptr;

PROD_INSTANCE
temp_p,
Get_Id(),
IsNull(),
IdSetSize(),
FirstElement(),
IdSetTail();

char
*edge_name;

first_x =0;
if (h != NULL)
{
input_count = IntValue(IdSetSize(h->inh_input_id_set));
output_count = IntValue(IdSetSize(h->inh_output_id_set));
state_count = IntValue(IdSetSize(h->state_id_set));

if (input_count > 0)
{

  /* add the new vertex */
  first_x = 100;
  node_x = first_x + (80 * (input_count - 1) / 2);

  k = Vtx_Name_Is_In_List("", "INPUT", "", h->operator_list);
  if (k != NULL)
  {
    k->op->is_deleted = FALSE;
    k->op->x = node_x;
    k->op->name_x = node_x;
    k->op->met_x = node_x;
  }
  else
  {
    new_vertex = Make_Operator_Node("INPUT",
                 "",
                 "INPUT",
                 "",
                 Get_Unique_Id(),
                 0,
                 node_x,
                 620,
                 2,
                 fake_color,
                 fake_font,
                 node_x,
                 640,
                 fake_font,
                 node_x,
                 640,
                 FALSE,
                 FALSE,
                 FALSE,
                 FALSE,
                 FALSE);

    Link_Operator(new_vertex, h);
    k = new_vertex;
  }

  /* add the new edgess */
  edge_x = first_x;
  edge_label_y = 555;
  temp_p = h->inh_input_id_set;
  while (!BoolValue(IsNull(temp_p)))
  {
    edge_name = str0_to_str_ro(StrValue(Get_Id(FirstElement(temp_p))));
    temp_p = IdSetTail(temp_p);

    t = Edge_Name_Is_In_List(edge_name, NULL, k, h->stream_list);
    if (t != NULL)
    {
      t->is_deleted = FALSE;
      t->name_x = edge_x - 10;
      t->name_y = edge_label_y;
      t->arc->x = edge_x;
      t->latency_x = edge_x - 10;
    }
    else
    {
      spl_ptr = (SPLINE_PTR) malloc(sizeof(SPLINE_NODE));

      spl_ptr->next = NULL;
      spl_ptr->x = edge_x;
      spl_ptr->y = 500;

      new_edge = Make_Stream_Node(edge_name,
                 Get_Unique_Id(),
                 NULL,
                 k,
                 fake_font,
                 edge_x - 10,
                 edge_label_y,
                 fake_font,
                 edge_x - 10,
                 edge_label_y,
                 spl_ptr,
                 fake_latency,
                 FALSE,
                 FALSE,
                 FALSE,
                 FALSE);
      Link_Stream(new_edge, h);
    }

    edge_x = edge_x + 80;
    edge_label_y = edge_label_y + 10;
```

```
    }


}

if (output_count > 0)
{
  first_x = first_x + 80 * input_count + 100;
  node_x = first_x + (80 * (output_count - 1) / 2);

  k = Vtx_Name_Is_In_List("", "OUTPUT", "", h->operator_list);
  if (k != NULL)
  {
    k->op->is_deleted = FALSE;
    k->op->x = node_x;
    k->op->name_x = node_x;
    k->op->met_x = node_x;
  }
  else
  {
    new_vertex = Make_Operator_Node("OUTPUT",
                    "",
                    "OUTPUT",
                    "",
                    Get_Unique_Id(),
                    0,
                    node_x,
                    620,
                    2,
                    fake_color,
                    fake_font,
                    node_x,
                    640,
                    fake_font,
                    node_x,
                    640,
                    FALSE,
                    FALSE,
                    FALSE,
                    FALSE,
                    FALSE);

    Link_Operator(new_vertex, h);
    k = new_vertex;
  }

  /* add the new edgess */
  edge_x = first_x;
  edge_label_y = 555;
  temp_p = h->inh_output_id_set;
  while (!BoolValue(IsNull(temp_p)))
  {
    edge_name = str0_to_str_ro(StrValue(Get_Id(FirstElement(temp_p))));
    temp_p = IdSetTail(temp_p);

    t = Edge_Name_Is_In_List(edge_name, k, NULL, h->stream_list);
    if (t != NULL)
    {
      t->is_deleted = FALSE;
      t->name_x = edge_x - 10;
      t->arc->x = edge_x;
      t->latency_x = edge_x - 10;
    }
    else
    {
      spl_ptr = (SPLINE_PTR) malloc(sizeof(SPLINE_NODE));

      spl_ptr->next = NULL;
      spl_ptr->x = edge_x;
      spl_ptr->y = 500;

      new_edge = Make_Stream_Node(edge_name,
                    Get_Unique_Id(),
                    k,
                    NULL,
                    fake_font,
                    edge_x - 10,
                    edge_label_y,
                    fake_font,
                    edge_x - 10,
                    edge_label_y,
                    spl_ptr,
                    fake_latency,
                    FALSE,
                    FALSE,
                    FALSE,
                    FALSE);
      Link_Stream(new_edge, h);
    }

    edge_x = edge_x + 80;
    edge_label_y = edge_label_y + 10;
  }

}

if (state_count > 0)
{
  first_x = first_x + 80 * output_count + 100;
  node_x = first_x + (80 * (state_count - 1) / 2);

  k = Vtx_Name_Is_In_List("", "STATE", "", h->operator_list);
  if (k != NULL)
  {
    k->op->is_deleted = FALSE;
    k->op->x = node_x;
    k->op->name_x = node_x;
    k->op->met_x = node_x;
  }
  else
  {
    new_vertex = Make_Operator_Node("STATE",
                    "",
                    "STATE",
                    "",
                    Get_Unique_Id(),
                    0,
                    node_x,
                    620,
                    2,
                    fake_color,
                    fake_font,
                    node_x,
                    640,
                    fake_font,
                    node_x,
```

```
                    640,
                    FALSE,
                    FALSE,
                    FALSE,
                    FALSE,
                    FALSE);

    Link_Operator(new_vertex, h);
    k = new_vertex;
  }

  /* add the new edgess */
  edge_x = first_x;
  edge_label_y = 555;
  temp_p = h->state_id_set;
  while (!BoolValue(IsNull(temp_p)))
  {
    edge_name = str0_to_str_ro(StrValue(Get_Id(FirstElement(temp_p))));
    temp_p = IdSetTail(temp_p);

    t = Edge_Name_Is_In_List(edge_name, k, k, h->stream_list);
    if (t != NULL)
    {
      t->is_deleted = FALSE;
      t->name_x = edge_x - 10;
      t->arc->x = edge_x - 5;
      t->arc->next->x = edge_x + 5;
      t->latency_x = edge_x - 10;
    }
    else
    {
      spl_ptr = (SPLINE_PTR) malloc(sizeof(SPLINE_NODE));
      spl_ptr->next = (SPLINE_PTR) malloc(sizeof(SPLINE_NODE));

      spl_ptr->x = edge_x - 5;
      spl_ptr->y = 550;
      spl_ptr->next->x = edge_x + 5;
      spl_ptr->next->y = 550;
      spl_ptr->next->next = NULL;

      new_edge = Make_Stream_Node(edge_name,
                    Get_Unique_Id(),
                    k,
                    k,
                    fake_font,
                    edge_x - 10,
                    edge_label_y,
                    fake_font,
                    edge_x - 10,
                    edge_label_y,
                    spl_ptr,
                    fake_latency,
                    FALSE,
                    FALSE,
                    FALSE,
                    TRUE);
      Link_Stream(new_edge, h);
    }

    edge_x = edge_x + 80;
    edge_label_y = edge_label_y + 10;
  }
}
```

```
  }

}

/*-------------------------------------------------------------------------*/
void Remove_Input_Output_State_Nodes(h)
  HeadPtr h;
{
  OPNodePTR
  Vtx_Name_Is_In_List(),
  k;

  STREAM
  Edge_Name_Is_In_List(),
  t;

  PROD_INSTANCE
  temp_p,
  Get_Id(),
  IsNull(),
  FirstElement(),
  IdSetTail();

  char
  *edge_name;

  if (h != NULL)
  {
  k = Vtx_Name_Is_In_List("", "INPUT", "", h->operator_list);
  if (k != NULL)
  {
    k->op->is_deleted = TRUE;
    temp_p = h->inh_input_id_set;
    while (!BoolValue(IsNull(temp_p)))
    {
      edge_name = str0_to_str_ro(StrValue(Get_Id(FirstElement(temp_p))));
      temp_p = IdSetTail(temp_p);

      t = Edge_Name_Is_In_List(edge_name, NULL, k, h->stream_list);
      if (t != NULL)
      {
        t->is_deleted = TRUE;
      }
    }
  }

  k = Vtx_Name_Is_In_List("", "OUTPUT", "", h->operator_list);
  if (k != NULL)
  {
    k->op->is_deleted = TRUE;
    temp_p = h->inh_output_id_set;
    while (!BoolValue(IsNull(temp_p)))
    {
      edge_name = str0_to_str_ro(StrValue(Get_Id(FirstElement(temp_p))));
      temp_p = IdSetTail(temp_p);

      t = Edge_Name_Is_In_List(edge_name, k, NULL, h->stream_list);
      if (t != NULL)
      {
        t->is_deleted = TRUE;
      }
    }
```

```
  }

  k = Vtx_Name_Is_In_List("", "STATE", "", h->operator_list);
  if (k != NULL)
  {
    k->op->is_deleted = TRUE;
    temp_p = h->state_id_set;
    while (!BoolValue(IsNull(temp_p)))
    {
      edge_name = str0_to_str_ro(StrValue(Get_Id(FirstElement(temp_p))));
      temp_p = IdSetTail(temp_p);

      t = Edge_Name_Is_In_List(edge_name, k, k, h->stream_list);
      if (t != NULL)
      {
        t->is_deleted = TRUE;
      }
    }
  }

  }
}
/*-------------------------------------------------------------------------*/
void Remove_Deleted_Operators(h)
  HeadPtr h;
{
  OPNodePTR
  temp,
  k;

  k = h->operator_list;

  while (k != NULL)
  {
  if (k->op->is_deleted)
  {
    h->operator_list = k->next;
    free(k->op->name);
    free(k->op->optional_type_name);
    free(k->op->oper_name);
    free(k->op->parameter_list);
    free(k->op);
    free(k);
    k = h->operator_list;
  }
  else
  {
    k = NULL;
  }
  }

  if (h->operator_list != NULL)
  {
  k = h->operator_list;
  while (k->next != NULL)
  {
    if (k->next->op->is_deleted)
    {
      temp = k->next;
      k->next = temp->next;
      free(temp->op->name);
      free(temp->op->optional_type_name);
      free(temp->op->oper_name);
      free(temp->op->parameter_list);
      free(temp->op);
      free(temp);
    }
    else
    {
      k = k->next;
    }

  }
  }

}

/*-------------------------------------------------------------------------*/
void Remove_Deleted_Streams(h)
  HeadPtr h;
{
  ST_PTR
  temp,
  t;

  int
  WipeOutSpline();

  t = h->stream_list;

  while (t != NULL)
  {
  if (t->st->is_deleted)
  {
    h->stream_list = t->next;
    free(t->st->name);
    WipeOutSpline(t->st->arc);
    free(t->st);
    free(t);
    t = h->stream_list;
  }
  else
  {
    t = NULL;
  }
  }

  if (h->stream_list != NULL)
  {
  t = h->stream_list;
  while (t->next != NULL)
  {
    if (t->next->st->is_deleted)
    {
      temp = t->next;
      t->next = temp->next;
      free(temp->st->name);
      WipeOutSpline(temp->st->arc);
      free(temp->st);
      free(temp);
    }
    else
    {
      t = t->next;
    }
```

```
  }
  }

}
/*-------------------------------------------------------------------------*/
void Restore_Deleted_Operators(h)
  HeadPtr h;
{
  OPNodePTR
  k;

  k = h->operator_list;

  while (k != NULL)
  {
  k->op->is_deleted = false;
  k->op->is_modified = false;
  k->op->is_new = false;
  k = k->next;
  }
}
/*-------------------------------------------------------------------------*/
void Restore_Deleted_Streams(h)
  HeadPtr h;
{
  ST_PTR
  t;

  t = h->stream_list;

  while (t != NULL)
  {
  t->st->is_deleted = false;
  t->st->is_modified = false;
  t->st->is_new = false;
  t = t->next;
  }
}

/*-------------------------------------------------------------------------*/
int Get_Met(v)
  PROD_INSTANCE
  v;
{
  PROD_INSTANCE
  t,
  Get_Vertex_Time(),
  Convert_Time_To_Integer();

  t = Get_Vertex_Time(v);

  return(IntValue(Convert_Time_To_Integer(t)));
}
/*-------------------------------------------------------------------------*/
int Get_Latency(e)
  PROD_INSTANCE
  e;
{
  PROD_INSTANCE
  t,
  Get_Edge_Time(),
  Convert_Time_To_Integer();

  t = Get_Edge_Time(e);

  return(IntValue(Convert_Time_To_Integer(t)));
}
/*-------------------------------------------------------------------------*/
TYPE_LIST Find_Type_Node_from_Data(p)
  PROD_INSTANCE p;
{
  char*name;

  TYPE_LIST
  temp_list;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Find_Type_Node_from_Data\n");
#endif

  name = str0_to_str_ro(StrValue(Get_Id(id_from_Data(p))));
  temp_list = Global_Type_List;
  while (temp_list != NULL)
  {
  if (strcmp(name, temp_list->type_name) == 0)
    return(temp_list);
  else
    temp_list = temp_list->next;
  }
  return(NULL);
}

/*-------------------------------------------------------------------------*/
HeadPtr Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p)
        PROD_INSTANCE p;
{
  int   prod_no;

  HeadPtr
        h,
        Op_Number_Is_In_List(),
        Op_Name_Is_In_List();

  char
        *name;

  PROD_INSTANCE
  Get_Op_Name();

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec\n");
#endif

  prod_no =(int)p;
  h = Op_Number_Is_In_List(prod_no);

  if (h == NULL)
  {
        /* since op_id_no corresponding to the Op production
        instance may have changed since last update to the
        op_node, need to check one more time for op_node with
        matching name */
```

```
#ifdef SDE_DEBUG_2
        /* debugging */
        printf("Op_node id may have changed\n");
#endif

  if (production(p) == prod_op)
        name = str0_to_str_ro(StrValue(Get_Id(id_from_Op(p))));
  else if (production(p) == prod_t_op_impl)
        name = str0_to_str_ro(StrValue(Get_Id(id_from_TOpImpl(p))));
  else
        name = str0_to_str_ro(StrValue(Get_Id(id_from_TOpSpec(p))));



        if (name != NULL)
                h = Op_Name_Is_In_List(name);

        if (h != NULL)
        {
                /* update id of op_node */
                h->prod_no = prod_no;
        }
  }


  return(h);
}
/*-------------------------------------------------------------------------*/
void Move_To_Structure_Front(p)
         HeadPtr p;
{
  extern HeadPtr
        prototype;

HeadPtr
        Curr,
        Prev;

#ifdef GRAPHICS_DEBUG
  /* debugging */
  printf("Entering Move_To_Structure_Front\n");
#endif

  Prev = NULL;
  Curr = prototype;

  /* search for p and remove it from the "prototype" list if found */
  while (Curr != NULL)
  {
        if (Curr == p)
        {
                if (Prev == NULL)
                {
                        prototype = Curr->next;
                }
                else
                {
                        Prev->next = Curr->next;
                }
                Curr = NULL; /* to get out of the while loop */
        }
        else
        {
                Prev = Curr;
                Curr = Curr->next;
        }
  }

  /* reinsert p to front of the "prototype" list */
  p->next = prototype;
  prototype = p;
}

/*-------------------------------------------------------------------------*/
PROD_INSTANCE Get_Id_From_Inputs_List(p)
  PROD_INSTANCE p;
{

  PROD_INSTANCE
  temp_p,
  Get_Id_From_Type_Decl();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Get_Id_From_Inputs_List\n");
#endif

  temp_p = p;
  while ((production(temp_p) != prod_input_list) &&
    (production(temp_p) != prod_inputs) &&
    (production(temp_p) != prod_type_decl))
  {
  temp_p = father(temp_p);
  }

  if (production(temp_p) == prod_input_list)
  {
#ifdef SDE_DEBUG_2
  /*debugging*/
  printf("Get_Id_From_Inputs_List: production(temp_p) == prod_input_list\n");
#endif

/*
  temp_p = son(temp_p, 1);
*/
  temp_p = o_inputs_from_InputsListPair(temp_p);
  }

  if (production(temp_p) == prod_inputs)
  {
#ifdef SDE_DEBUG_2
  /*debugging*/
  printf("Get_Id_From_Inputs_List: production(temp_p) == prod_inputs\n");
#endif

/*
  temp_p = son(temp_p, 1);
*/
  temp_p = type_declarations_from_OpInputs(temp_p);
  }

  if (production(temp_p) == prod_type_decl)
  {
#ifdef SDE_DEBUG_2
  /*debugging*/
```

```
  printf("Get_Id_From_Inputs_List: production(temp_p) == prod_type_decl\n");
#endif

  return(Get_Id_From_Type_Decl(temp_p));
  }
  else
  {
  /* production(temp_p) == OpInputsNone */
  return(Make_IdNull());
  }

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Leaving Get_Id_From_Inputs_List\n");
#endif
}
/*--------------------------------------------------------------------------*/
PROD_INSTANCE Get_Id_From_Outputs_List(p)
  PROD_INSTANCE p;
{

  PROD_INSTANCE
  temp_p,
  Get_Id_From_Type_Decl();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Get_Id_From_Outputs_List\n");
#endif

  temp_p = p;
  while ((production(temp_p) != prod_output_list) &&
     (production(temp_p) != prod_outputs) &&
     (production(temp_p) != prod_type_decl))
  {
  temp_p = father(temp_p);
  }

  if (production(temp_p) == prod_output_list)
/*
  temp_p = son(temp_p, 1);
*/
  temp_p = o_outputs_from_OutputsListPair(temp_p);

  if (production(temp_p) == prod_outputs)
  {
/*
  temp_p = son(temp_p, 1);
*/
  temp_p = type_declarations_from_OpOutputs(temp_p);
  }

  if (production(temp_p) == prod_type_decl)
  {
  return(Get_Id_From_Type_Decl(temp_p));
  }
  else
  {
  /* production(temp_p) == OpOutputsNone */
  return(Make_IdNull());
  }


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Leaving Get_Id_From_Outputs_List\n");
#endif
}
/*--------------------------------------------------------------------------*/
PROD_INSTANCE Get_Id_From_Stream(p)
  PROD_INSTANCE p;
{

  PROD_INSTANCE
  temp_p,
  Get_Id_From_Type_Decl();

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Get_Id_From_Stream\n");
#endif

  temp_p = p;
  while ((production(temp_p) != prod_stream) &&
     (production(temp_p) != prod_type_decl))
  {
  temp_p = father(temp_p);
  }

  if (production(temp_p) == prod_stream)
/*
  temp_p = son(temp_p, 1);
*/
  temp_p = type_declarations_from_Streams(temp_p);

  return(Get_Id_From_Type_Decl(temp_p));

#ifdef SDE_DEBUG_1
/* debugging */
  printf("Leaving Get_Id_From_Stream\n");
#endif
}
/*--------------------------------------------------------------------------*/
PROD_INSTANCE Get_Operator_Id_From_Constraints(p)
  PROD_INSTANCE p;
{

  PROD_INSTANCE
  temp_p,
  Make_IdNull();

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Get_Operator_Id_From_Constraints\n");
#endif

  temp_p = p;
  while ((production(temp_p) != prod_constraints) &&
     (production(temp_p) != prod_a_constraint))
  {
  temp_p = father(temp_p);
  }

  if (production(temp_p) == prod_constraints)
  temp_p = a_constraint_from_ConstraintsPair(temp_p);
```

```
  if (production(temp_p) == prod_a_constraint)
  {
  return(operator_id_from_AConstraint(temp_p));
  }
  else
  {
  return(OperatorIdNull);
  }

#ifdef SDE_DEBUG_1
/* debugging */
  printf("Leaving Get_Operator_Id_From_Constraints\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE Get_Id_From_Type_Decl(p)
  PROD_INSTANCE p;
{

  PROD_INSTANCE
  temp_p,
  Make_IdNull(),
  IsTypeDeclNil(),
  IsADeclNil(),
  IsIdListNil();

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Get_Id_From_Type_Decl\n");
#endif

  temp_p = p;

  if (BoolValue(IsTypeDeclNil(temp_p)))
  {
  return(Make_IdNull());
  }
  else
  {
/*
  temp_p = son(temp_p, 1);
*/
  temp_p = a_decl_from_TypeDeclPair(temp_p);
  }

  if (BoolValue(IsADeclNil(temp_p)))
  {
  return(Make_IdNull());
  }
  else
  {
/*
        temp_p = son(temp_p, 1);
*/
        temp_p = id_list_from_ADecl(temp_p);
  }

  if (BoolValue(IsIdListNil(temp_p)))
  {
  return(Make_IdNull());
  }
  else
  {
/*
  return(son(temp_p, 1));
*/
  return(id_from_IdPair(temp_p));
  }


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Leaving Get_Id_From_Type_Decl\n");
#endif
}
/*-------------------------------------------------------------------------*/
char *Get_Parameters_From_Operator_Id_Pairs(op_id_pairs)
  PROD_INSTANCE op_id_pairs;
{

  PROD_INSTANCE
  aid_list_1,
  aid_list_2;

  char
  *Get_Parameters_From_Alone_Id_List(),
  *list_1,
  *list_2,
  dummy[100];

  if (production(op_id_pairs) == prod_operator_id_pairs)
  {
  aid_list_1 = alone_id_list_1_from_OperatorIdPairs(op_id_pairs);
  aid_list_2 = alone_id_list_2_from_OperatorIdPairs(op_id_pairs);
  list_1 = Get_Parameters_From_Alone_Id_List(aid_list_1);
  list_2 = Get_Parameters_From_Alone_Id_List(aid_list_2);
  sprintf(dummy, "%s%s%s", list_1, "|", list_2);
/*
printf("Get_Parameters before free list: dummy = %s\n", dummy);
*/
  free(list_1);
  free(list_2);
/*
printf("Get_Parameters after free list: dummy = %s\n", dummy);
*/
  return(strdup(dummy));
  }
  else
  {
  return(strdup(""));
  }
}

/*-------------------------------------------------------------------------*/
char *Get_Parameters_From_Alone_Id_List(aidl)
  PROD_INSTANCE aidl;
{
  char
    *head_of_string,
    dummy[100];

  PROD_INSTANCE
    temp_p;

  temp_p = aidl;
```

```
  if (production(temp_p) != prod_alone_id_pair)
  {
    return(strdup(""));
  }
  else
  {

    sprintf(dummy, "%s", "");

    temp_p = aidl;

    while (production(temp_p) == prod_alone_id_pair)
    {
      strcat(dummy, ",");
      strcat(dummy, str0_to_str_ro(StrValue(Get_Id(
                id_from_AIdPair(temp_p)))));
      temp_p = alone_id_list_from_AIdPair(temp_p);
    }

/*
printf("Get_Parameters_From_Alone_Id_List: id_list = %s\n", dummy);
*/

    /* skip the first comma */
    head_of_string = dummy;
    while (*head_of_string != ',')
    {
      head_of_string ++;
    }

    return(strdup(head_of_string+1));
  }
}
%]

/*****************************************************************************
****  This fourth set of functions were already existing and
****  for use with the Graphic Editor.
****  Documented originally as file: ed.3.ss1
*****************************************************************************/

%[
OPNodePTR Vtx_Name_Is_In_List(optional_type_name, oper_name, parameter_list, p)
        OPNodePTR
          p;
        char
          *optional_type_name,
    *oper_name,
    *parameter_list;
{
  /* Compare "name" with the names stored in each of the operators in the
        list pointed to by "p".  If name is found, return a pointer to that
        node, otherwise, return NULL.
  */

  OPERATOR
        q;

  PROD_INSTANCE
  EqualOpId();

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Vtx_Name_Is_In_List\n");
#endif

#ifdef SDE_DEBUG_3
  printf("Vtx_Name_Is_In_List: optional_type_name = %s\n", optional_type_name);
  printf("Vtx_Name_Is_In_List: oper_name = %s\n", oper_name);
  printf("Vtx_Name_Is_In_List: parameter_list = %s\n", parameter_list);
#endif

  while (p != NULL)
        {
          q = p->op;
#ifdef SDE_DEBUG_3
printf("Vtx_Name_Is_In_List: oper_name = %s\n", q->oper_name);
#endif

    if (strcmp(q->name, "") != 0)
    {
          if (strcmp(oper_name, q->oper_name) == 0)
    {
      if (strcmp(optional_type_name, "") == 0)
      {
#ifdef SDE_DEBUG_3
printf("Vtx_Name_Is_In_List: found vertex\n");
#endif
        return(p);
      }
      else
      {
#ifdef SDE_DEBUG_3
printf("Vtx_Name_Is_In_List: optional_type_name = %s\n", q->optional_type_name);
printf("Vtx_Name_Is_In_List: parameter_list  = %s\n", q->parameter_list);
#endif
        if (
            (strcmp(optional_type_name, q->optional_type_name) == 0)
            && (strcmp(parameter_list, q->parameter_list) == 0))

        {
#ifdef SDE_DEBUG_3
printf("Vtx_Name_Is_In_List: found vertex\n");
#endif
            return(p);
        }
        else
            p = p->next;
      }
    }
    else
    {
              p = p->next;
    }
    }
    else
    {
    p = p->next;
    }
       }
#ifdef SDE_DEBUG_3
printf("Vtx_Name_Is_In_List: can't find vertex\n");
#endif
  return(NULL);
}
/*---------------------------------------------------------------------------*/
```

```
STREAM Edge_Name_Is_In_List(edge_name, from, to, p)
        char
          *edge_name;

  OPNodePTR
    from,
    to;

        ST_PTR
           p;
{
  STREAM
        q;


#ifdef GRAPHICS_DEBUG
  /* debugging */
  printf("Entering Edge_Name_Is_In_List\n");
#endif

#ifdef SDE_DEBUG_3
printf("Edge_Name_Is_In_List: edge_name = %s\n", edge_name);
printf("Edge_Name_Is_In_List: from->name = %s\n", (from == NULL)?"EXTERNAL":from->op-
>name);
printf("Edge_Name_Is_In_List: from = %d\n", from);
printf("Edge_Name_Is_In_List: to->name = %s\n", (to == NULL)?"EXTERNAL":to->op->name);
printf("Edge_Name_Is_In_List: to = %d\n", to);
#endif

  while (p != NULL)
        {
        q = p->st;

#ifdef SDE_DEBUG_3
printf("Edge_Name_Is_In_List: q->name = %s\n", q->name);

printf("Edge_Name_Is_In_List: q->from->name = %s\n", (q->from == NULL)?"EXTERNAL":q-
>from->op->name);
printf("Edge_Name_Is_In_List: q->from = %d\n", (q->from));
printf("Edge_Name_Is_In_List: q->to->name = %s\n", (q->to == NULL)?"EXTERNAL":q->to-
>op->name);
printf("Edge_Name_Is_In_List: q->to = %d\n", (q->to));
#endif

          if ((strcmp(edge_name, q->name) == 0))
          {

      if (from == q->from && to == q->to)
      {
#ifdef SDE_DEBUG_3
    printf("Edge_Name_Is_In_List: find edge\n");
#endif
              return(q);
           }
      else
      {
        p = p->next;
      }
    }
    else
    {
         p = p->next;
    }
        }

#ifdef SDE_DEBUG_3
printf("Edge_Name_Is_In_List: can't find edge\n");
#endif


  return(NULL);
);

/*-------------------------------------------------------------------------------*/
HeadPtr Op_Name_Is_In_List(my_oper_name)
         char *my_oper_name;
{
  extern HeadPtr
        prototype;

  HeadPtr p;

#ifdef GRAPHICS_DEBUG
  /* debugging */
  printf("Entering Op_Name_Is_In_List\n");
#endif

  p = prototype;

  while (p != NULL)
        {
          if (strcmp(p->name, my_oper_name) == 0)
                {
                  return(p);
                }
          else
                p = p->next;
        }
  return(p);
}
/*-------------------------------------------------------------------------------*/
HeadPtr Op_Number_Is_In_List(prod_no)
  int
  prod_no;
{
  extern HeadPtr
        prototype;

  HeadPtr p;

#ifdef GRAPHICS_DEBUG
  /* debugging */
  printf("Entering Op_Number_Is_In_List\n");
#endif

  p = prototype;
  while (p != NULL)
        {
          if (p->prod_no == prod_no)
                {
                  return(p);
                }
          else
                p = p->next;
        }
  return(p);
```

```
)

/*-------------------------------------------------------------------------*/
term_sys()
{
  void
  save_graphic_file();

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering term_sys\n");
#endif


  kill_viewer();

}
/*-------------------------------------------------------------------------*/
void save_graphic_file()
{

  void
        Output_OperatorS_Operators(),
        Output_OperatorS_Streams();

  FILE
        *fp, *fopen();

  char
        buffer[30],
        *file_name;

  PROD_INSTANCE
  temp_p,
  temp_op_impl_list,
  temp_t_op_impl,
  ops;

  HeadPtr
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(),
        p;

  extern HeadPtr
        prototype;


  /*
        The objective is to search the data structure(s) and create graphic attribute
        information for each of the objects found.
 */


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering save_graphic_file\n");
#endif

  fp = fopen("attr_file_name.grf", "r");
  fscanf(fp, "%s", buffer);
  fscanf(fp, "%s", buffer);
  file_name = (char *)malloc(strlen(buffer));
  fclose(fp);
  strcpy(file_name, buffer);

  fp = fopen(file_name, "w");

  ops = psdl_components_from_Prot(Global_Proto);

  while (production(ops) == prod_psdl_pair)
        {
    temp_p = component_from_PsdlPair(ops);
    if (production(temp_p) == prod_op)
    {

#ifdef SDE_DEBUG_3
printf("save_graphic_file: operator id = %s\n",
str0_to_str_ro(StrValue(Get_Id(id_from_Op(temp_p)))));
#endif
    p = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(temp_p);
    if (p != NULL)
    {

          fprintf(fp, "OPERATOR\n");
          fprintf(fp, "%s\n", p->name);
          Output_OperatorS_Operators(fp, p->operator_list);
          Output_OperatorS_Streams(fp, p->stream_list);
          fprintf(fp, "ENDOPERATOR\n");
    }
    }
    else if (production(temp_p) == prod_data)
    {
    if (production(type_impl_from_Data(temp_p)) == prod_type_impl)
    {
      temp_op_impl_list = operator_impl_list_from_TypeImpl(
                type_impl_from_Data(temp_p));
      while (production(temp_op_impl_list) == prod_op_impl_list_pair)
      {
        temp_t_op_impl = t_oper_impl_from_OpImplListPair(temp_op_impl_list);

        if (production(temp_t_op_impl) == prod_t_op_impl)
        {
#ifdef SDE_DEBUG_3
printf("save_graphic_file: operator id = %s\n",
str0_to_str_ro(StrValue(Get_Id(id_from_TOpImpl(temp_t_op_impl)))));
#endif
            p = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(temp_t_op_impl);
            if (p != NULL)
            {
                fprintf(fp, "OPERATOR\n");
                      fprintf(fp, "%s\n", p->name);
                      Output_OperatorS_Operators(fp, p->operator_list);
                      Output_OperatorS_Streams(fp, p->stream_list);
                      fprintf(fp, "ENDOPERATOR\n");
                }
        }

        temp_op_impl_list = operator_impl_list_from_OpImplListPair(

temp_op_impl_list);
      }
    }
    }

          ops = psdl_components_from_PsdlPair(ops);
        }
  fclose(fp);
```

```
#ifdef SDE_DEBUG_3
  printf("GRAPHIC INFORMATION SAVED\n");
#endif

}
/*-------------------------------------------------------------------------*/
void Output_OperatorS_Operators(fp, operator_list)
         FILE
           *fp;
         OPNodePTR
           operator_list;
{
  OPERATOR
        q;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Output_OperatorS_Operators\n");
#endif

  fprintf(fp, "OPERATORS\n");

  while (operator_list != NULL)
        {
          q = operator_list->op;

    if (!(q->is_deleted))
    {
          fprintf(fp, "%s\n", q->name);
          fprintf(fp, "%d\n", q->x);
          fprintf(fp, "%d\n", q->y);
          fprintf(fp, "%d\n", q->radius);
          fprintf(fp, "%d\n", q->color);
          fprintf(fp, "%d\n", q->name_font);
          fprintf(fp, "%d\n", q->name_x);
          fprintf(fp, "%d\n", q->name_y);
          fprintf(fp, "%d\n", q->met_font);
          fprintf(fp, "%d\n", q->met_x);
          fprintf(fp, "%d\n", q->met_y);
          fprintf(fp, "%s\n", (q->is_terminator==1?"TRUE":"FALSE"));
    }

          operator_list = operator_list->next;
        }
}
/*-------------------------------------------------------------------------*/
void Output_OperatorS_Streams(fp, stream_list)
         FILE
           *fp;
         ST_PTR
           stream_list;

{

  STREAM
        q;
  SPLINE_PTR
        r;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Output_OperatorS_Streams\n");
#endif

  fprintf(fp, "STREAMS\n");
  while (stream_list != NULL)
        {
          q = stream_list->st;

    if (!(q->is_deleted))
    {
          fprintf(fp, "%s\n", q->name);
          fprintf(fp, "%s\n", (q->from == NULL?"<NULL>":q->from->op->name));
          fprintf(fp, "%s\n", (q->to == NULL?"<NULL>":q->to->op->name));

          if ((q->from != NULL) || (q->to != NULL))
                {
                  fprintf(fp, "SPLINE\n");

                  if (q->arc != NULL)
                        {
                          r = q->arc;
                          while (r != NULL)
                                {
                                  fprintf(fp, "%d\n", r->x);
                                  fprintf(fp, "%d\n", r->y);
                                  r = r->next;
                                }
                        }
                  fprintf(fp, "SPLINEEND\n");
                }

    fprintf(fp, "%d\n", q->name_font);
    fprintf(fp, "%d\n", q->name_x);
    fprintf(fp, "%d\n", q->name_y);
    fprintf(fp, "%d\n", q->latency_font);
    fprintf(fp, "%d\n", q->latency_x);
    fprintf(fp, "%d\n", q->latency_y);

          fprintf(fp, "%s\n", (q->is_state_variable==1?"TRUE":"FALSE"));

    }

          stream_list = stream_list->next;
        }
}
/*-------------------------------------------------------------------------*/
BOOL Process_File()
{
  BOOL
        result,
        Get_Operator_list(),
        Get_Stream_List();

  HeadPtr
        header,
        Get_Header();

  FILE
        *fopen(),
        *fp;

  int
        Print_OperatorS_Operators(),
        Print_OperatorS_Streams(),
```

```
        Output_Structure(),
        n;

  char
        buffer[30],
        *file_name;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Process_File\n");
#endif

  /* Load file name from where graphic attributes will be read */

  fp = fopen("attr_file_name.grf", "r");
  fscanf(fp, "%s", buffer);

#ifdef SDE_DEBUG_2
  printf("%s\n", buffer);
#endif

  file_name = (char *)malloc(strlen(buffer));
  strcpy(file_name, buffer);

  if (strcmp(file_name, "new") != 0)
  {
  fp = fopen(file_name, "r");

        /*  consume the "OPERATOR" word and continue or finish */
        while ((n=fscanf(fp, "%s", buffer)) != EOF)
        {
        /* process an operator header */
                header = Get_Header(fp);
    if (header == NULL)
    {
      printf("Process_File: Get_Header FAILED\n");
      fclose(fp);
      return(FALSE);
    }

                /*process operator list*/
                result = Get_Operator_List(fp, header);
                if (result == FALSE)
                {
                        printf("Process_File: Get_Operator_list FAILED\n");
                        fclose(fp);
                        return(FALSE);
                }

                /*process stream list */
                result = Get_Stream_List(fp, header);
                if (result == FALSE)
                {
                        printf("Process_File: Get_Stream_List FALIED\n");
                        fclose(fp);
                        return(FALSE);
                }
        }

        fclose(fp);
        free(file_name);

        /* the following command is commented out for
         ease of debugging 1/21/94 */
        /* Output_Structure(); */
        return(TRUE);
  }
  else
  {
        return(TRUE);
  }
}
/*-------------------------------------------------------------------------*/
HeadPtr Get_Header(fp)
  FILE *fp;
{
  HeadPtr
        Op_Name_Is_In_List(),
  Make_Operator_Header(),
        p;

  void
  Link_To_Structure();

  char
        buffer[30];

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Get_Header\n");
#endif

  fscanf(fp, "%s", buffer);

#ifdef SDE_DEBUG_2
  printf("%s\n", buffer);
#endif

  p = Op_Name_Is_In_List(buffer);

  if (p == NULL)
  {
#ifdef SDE_DEBUG_3
  /*debugging*/
    printf("Get_Header: header node with name %s not found, Aborted\n", buffer);
#endif

  /* need to make new header node */
        p = Make_Operator_Header(buffer, NULL, NULL, Get_Unique_Id(), 0, FALSE);
        Link_To_Structure(p);
  }

  return(p);


}
/*-------------------------------------------------------------------------*/
BOOL Get_Operator_List(fp, head_node)
         HeadPtr
           head_node;

         FILE
           *fp;
{
  OPNodePTR
  Make_Operator_Node(),
```

```
  new_vertex;

  char
        *front_part,
        *middle_part,
        *suffix_part,
        *get_vertex_type_name(),
        *get_vertex_oper_name(),
        *get_vertex_parameters(),
  *remove_blanks_from_string(),
  *clean_name,
        buffer[100],
        name[100];

  int
  name_font,
  name_x,
  name_y,
        met,
  met_font,
  met_x,
  met_y,
        radius,
        x,
        y,
        color;

  BOOL
        is_terminator;

  OPNodePTR
        n,
        Vtx_Name_Is_In_List();

  HeadPtr
        k,
        Op_Name_Is_In_List();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Get_Operator_List\n");
#endif

  /* eat away the OPERATORS word */
  fscanf(fp, "%s", buffer);

  while(fscanf(fp, "%s", name) && (strcmp(name, "STREAMS") != 0))
  {
          fscanf(fp, "%d", &x);
          fscanf(fp, "%d", &y);
          fscanf(fp, "%d", &radius);
          fscanf(fp, "%d", &color);
    fscanf(fp, "%d", &name_font);
    fscanf(fp, "%d", &name_x);
    fscanf(fp, "%d", &name_y);
    fscanf(fp, "%d", &met_font);
    fscanf(fp, "%d", &met_x);
    fscanf(fp, "%d", &met_y);

          fscanf(fp, "%s", buffer);
          is_terminator = (strcmp(buffer, "TRUE")==0);
    clean_name = remove_blanks_from_string(name);
    front_part = get_vertex_type_name(clean_name);
    middle_part = get_vertex_oper_name(clean_name);
    suffix_part = get_vertex_parameters(clean_name);

          n = Vtx_Name_Is_In_List(front_part, middle_part, suffix_part, head_node-
>operator_list);
          if (n != NULL)
          {
          printf("Get_Operator_List: duplicated vertex node with name %s\n", name);
            return(FALSE);
          }
          else
    {
    /* need to create a new vertex_node */
           new_vertex = Make_Operator_Node(clean_name,
              front_part,
            middle_part,
            suffix_part,
                                        Get_Unique_Id(),
                                        NULL,
                                        x,
                                        y,
                                        radius,
                                        color,
                                        name_font,
                                        name_x,
                                        name_y,
                                        met_font,
                                        met_x,
                                        met_y,
                                        fake_is_deleted,
                                        fake_is_new,
                                        fake_is_composite,
                                        is_terminator,
                                        fake_is_modified);
             Link_Operator(new_vertex, head_node);
           }
    free(clean_name);
    free(front_part);
    free(middle_part);
    free(suffix_part);
  }

  return(TRUE);
}
/*-------------------------------------------------------------------------------*/
BOOL Get_Stream_List(fp, head_node)
          FILE
            *fp;
          HeadPtr
            head_node;
{
  int
  name_font,
  name_x,
  name_y,
  latency_font,
  latency_x,
  latency_y,
        latency;

  OPNodePTR
```

```
        from,
        to,
  Vtx_Op_Id_Is_In_List(),
        op_list;

  ST_PTR
        Make_Stream_Node(),
  new_edge,
        st;

  STREAM
        n,
  Edge_Name_Is_In_List();

  char
  *optional_type_name,
  *oper_name,
  *parameter_list,
        *get_vertex_type_name(),
        *get_vertex_oper_name(),
        *get_vertex_parameters(),
  *remove_blanks_from_string(),
  *clean_name,
        buffer[40],
        name[30];

  SPLINE_PTR
        GetSpline(),
        t;

  ST_PTR
        sp;
  BOOL
        is_state_variable;


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Get_Stream_List\n");
#endif

  op_list = head_node->operator_list;

  while (fscanf(fp, "%s", name) && (strcmp(name, "ENDOPERATOR") != 0))
        {

#ifdef SDE_DEBUG_3
printf("Get_Stream_List: stream name = %s\n", name);
#endif

          fscanf(fp, "%s", buffer);
          clean_name = remove_blanks_from_string(buffer);

          if (strcmp(clean_name, "<NULL>") == 0)
                {
#ifdef SDE_DEBUG_3
printf("Get_Stream_List: from node = %s\n", clean_name);
#endif

                  from = NULL;
                }
          else
                {
            optional_type_name = get_vertex_type_name(clean_name);

#ifdef SDE_DEBUG_3
printf("Get_Stream_List: from node = %s\n", clean_name);
printf("Get_Stream_List: optional_type_name = %s\n", optional_type_name);
#endif

            oper_name = get_vertex_oper_name(clean_name);

#ifdef SDE_DEBUG_3
printf("Get_Stream_List: from node = %s\n", clean_name);
printf("Get_Stream_List: oper_name = %s\n", oper_name);
#endif

            parameter_list = get_vertex_parameters(clean_name);

#ifdef SDE_DEBUG_3
printf("Get_Stream_List: from node = %s\n", clean_name);
printf("Get_Stream_List: parameter_list = %s\n", parameter_list);
#endif


                  from = Vtx_Name_Is_In_List(optional_type_name, oper_name,
                parameter_list, op_list);

                  free(clean_name);
      free(optional_type_name);
      free(oper_name);
      free(parameter_list);

                  if (from == NULL)
                        {
                          printf("Get_Stream_List: from vertex %s not found\n",
clean_name);
                          return (FALSE);
                        }
                }


          fscanf(fp, "%s", buffer);
          clean_name = remove_blanks_from_string(buffer);
          if (strcmp(clean_name, "<NULL>") == 0)
                {

#ifdef SDE_DEBUG_3
printf("Get_Stream_List: to node = %s\n", clean_name);
#endif

                  to = NULL;
                }
          else
                {
            optional_type_name = get_vertex_type_name(clean_name);

#ifdef SDE_DEBUG_3
printf("Get_Stream_List: to node = %s\n", clean_name);
printf("Get_Stream_List: optional_type_name = %s\n", optional_type_name);
#endif

            oper_name = get_vertex_oper_name(clean_name);

#ifdef SDE_DEBUG_3
printf("Get_Stream_List: to node = %s\n", clean_name);
```

```
printf("Get_Stream_List: oper_name = %s\n", oper_name);
#endif

            parameter_list = get_vertex_parameters(clean_name);

#ifdef SDE_DEBUG_3
printf("Get_Stream_List: to node = %s\n", clean_name);
printf("Get_Stream_List: parameter_list = %s\n", parameter_list);
#endif

                  to = Vtx_Name_Is_In_List(optional_type_name, oper_name,
                 parameter_list, op_list);

       free(clean_name);
       free(optional_type_name);
       free(oper_name);
       free(parameter_list);
                  if (to == NULL)
                        {
                          printf("Get_Stream_List: to vertex %s not found\n", buffer);
                          return(FALSE);
                        }
                }

          t = GetSpline(fp);

     fscanf(fp, "%d", &name_font);
     fscanf(fp, "%d", &name_x);
     fscanf(fp, "%d", &name_y);
     fscanf(fp, "%d", &latency_font);
     fscanf(fp, "%d", &latency_x);
     fscanf(fp, "%d", &latency_y);

          fscanf(fp, "%s", buffer);
          is_state_variable = (strcmp(buffer, "TRUE")==0);

          n = Edge_Name_Is_In_List(name, from, to, head_node->stream_list);
          if (n != NULL)
          {
            printf("Get_Stream_List: duplicated stream with name %s\n",
                                                                  name);
            return(FALSE);
          }
          else
          {
            /* create new stream node */
            new_edge = Make_Stream_Node(name,
                                Get_Unique_Id(),
                                from,
                                to,
             name_font,
             name_x,
             name_y,
                                      latency_font,
                                      latency_x,
                                      latency_y,
                                t,
                                fake_latency,
                                fake_is_deleted,
                                fake_is_new,
                                fake_is_modified,
                                fake_is_state
                                );
              Link_Stream(new_edge, head_node);
            }

          }

  return(TRUE);
}
/*---------------------------------------------------------------------------*/
SPLINE_PTR GetSpline(fp)
         FILE *fp;
{
  SPLINE_PTR
        dummy, last, p;

  char
        buffer[30];

  int
        x, y;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering GetSpline\n");
#endif

  dummy = (SPLINE_PTR)malloc(sizeof(SPLINE_NODE));
  dummy->next = NULL;
  last = dummy;

  /* eat away the SPLINE word */
  fscanf(fp, "%s", buffer);

  while(fscanf(fp, "%s", buffer) && (strcmp(buffer, "SPLINEEND") != 0))
        {
          sscanf(buffer, "%d", &x);
          fscanf(fp, "%d", &y);

          p = (SPLINE_PTR)malloc(sizeof(SPLINE_NODE));
          p->x = x;
          p->y = y;

          p->next = NULL;
          last->next = p;
          last = p;
        }
  last = dummy->next;
  free(dummy);
  return(last);
}
/*---------------------------------------------------------------------------*/
ST_PTR Make_Stream_Node(name, st_no, from, to,
       name_font, name_x, name_y,
       latency_font, latency_x, latency_y,
       spl, latency, is_deleted, is_new,
                        is_modified, is_state)
          char
            *name;
          OPNodePTR
            from,
            to;
          int
            name_font,
            name_x,
```

```
          name_y,
          latency_font,
          latency_x,
          latency_y,
          latency;
        BOOL
          is_deleted,
          is_new,
          is_modified,
          is_state;
        SPLINE_PTR
          spl;
{
  ST_PTR
       s;
  STREAM
       r;

#ifdef GRAPHICS_DEBUG
  /* debugging */
  printf("Entering Make_Stream_Node\n");
#endif

  s = (ST_PTR)malloc(sizeof(ST_HEAD));
  r = (STREAM)malloc(sizeof(ST_NODE));
  r->name = strdup(name);
  r->id = st_no;
  r->from = from;
  r->to = to;
      /* r->from and r->to now points to the op_nodes
        which constain the info of the two end-points
        of the data stream */
  r->arc = spl;
  r->latency = latency;
  r->is_state_variable = is_state;
  r->is_new = is_new;
  r->is_deleted = is_deleted;
  r->is_modified = is_modified;

  r->name_font = name_font;
  r->name_x = name_x;
  r->name_y = name_y;

  r->latency_font = latency_font;
  r->latency_x = latency_x;
  r->latency_y = latency_y;

  s->st = r;
  s->next = NULL;
  return(s);
}
/*-------------------------------------------------------------------------*/
void Link_To_Structure(p)
       HeadPtr p;
{
  extern HeadPtr
       prototype;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Link_To_Structure\n");
#endif

  p->next = prototype;
  prototype = p;
}
/*-------------------------------------------------------------------------*/
int Get_Unique_Id()
{
  extern int
  unique_id_count;

  int temp;

  temp = unique_id_count;
  unique_id_count = unique_id_count + 1;
  return(temp);
}
/*-------------------------------------------------------------------------*/
Output_Structure()
{
  int
       Print_PrototypeS_Operators(),
       Print_PrototypeS_Streams();

  extern HeadPtr
       prototype;

  HeadPtr
       p;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Output_Structure\n");
#endif

  printf("----------- STRUCTURE ------------\n");
  p = prototype;
  while (p != NULL)
       {
         printf("OPERATOR\n");
         printf("%s\n", p->name);
         printf("%d\n", p->op_id_no);
         printf("%s\n", (p->marked_for_delete == TRUE)?"TRUE":"FALSE");
         printf("%d\n", p->met);
         printf("%s\n", (p->is_composite==1?"IS_COMPOSITE":"NOT_COMPOSITE"));
         /* Print_OperatorS_Operators(p->operator_list); */
         /* Print_OperatorS_Streams(p->stream_list); */
         printf("ENDOPERATOR\n");
         p = p-> next;
       }
}
/*-------------------------------------------------------------------------*/
Print_OperatorS_Operators(o_list)
        OPNodePTR o_list;
{
  OPERATOR
       q;

#ifdef SDE_DEBUG_1
  printf("Entering Print_OperatorS_Operators\n");
#endif

  printf("OPERATORS\n");
  while (o_list != NULL)
       {
```

```
        q = o_list->op;

    printf("o_list = %d\n", (int)o_list);
    printf("o_list->op = %d\n", (int)q);
        printf("name: %s\n", q->name);
        printf("name_font: %d\n", q->name_font);
        printf("op_id: %d\n", q->id);
        printf("met: %d \n", q->met);
        printf("met_font: %d\n", q->met_font);
        printf("x: %d\n", q->x);
        printf("y: %d\n", q->y);
        printf("radius: %d\n", q->radius);
        printf("color: %d\n", q->color);
        printf("deleted: %s\n", (q->is_deleted==1?"TRUE":"FALSE"));
        printf("new: %s\n", (q->is_new==1?"TRUE":"FALSE"));
        printf("composite: %d\n", q->is_composite);
        printf("terminator: %s\n", (q->is_terminator==1?"TRUE":"FALSE"));
        printf("modified: %s\n", (q->is_modified==1?"TRUE":"FALSE"));

        o_list = o_list->next;
      }
}
/*-------------------------------------------------------------------------*/
Print_OperatorS_Streams(s_list)
        ST_PTR
          s_list;
{
  STREAM
      q;

  SPLINE_PTR
      r;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Print_OperatorS_Streams\n");
#endif

  printf("STREAMS\n");
  while (s_list != NULL)
      {
        q = s_list->st;

        printf("%s\n", q->name);
        printf("%d\n", q->id);
        printf("%s\n", (q->from == NULL?"<NULL>":q->from->op->name));
        printf("%s\n", (q->to == NULL?"<NULL>":q->to->op->name));

        printf("SPLINE\n");
        r = q->arc;
        while (r != NULL)
            {
              printf("%d\n", r->x);
              printf("%d\n", r->y);
              r = r->next;
            }

        printf("SPLINEEND\n");
        printf("%d\n", q->latency);
        printf("%s\n", (q->is_deleted==1?"TRUE":"FALSE"));
        printf("%s\n", (q->is_new==1?"TRUE":"FALSE"));
        printf("%s\n", (q->is_state_variable==1?"TRUE":"FALSE"));
        printf("%s\n", (q->is_modified==1?"TRUE":"FALSE"));

        s_list = s_list->next;
      }
}
%]
/*-------------------------------------------------------------------------*/

/*************************************************************************
****  This fifth set of functions were already existing and
****  designed to help enforce the consistency of PSDL program.
****  Documented originally as file: ed.4.ssl
*************************************************************************/
 /* this file contains all the routines needed to enforce consistency of the
    PSDL program */


/*-------------------------------------------------------------------------*/
/* The following definitions generate a Graphic Editor entry in the editor's
   main menu.  They are a faithful copy of what is stated in the synthesizer's
   user's manual.  Except that in the manual it is explained that a ".c" file
   with these definitions will do.  While investigating this, I arrived to the
   conclusion that such is not the case.  The definition must be included in
   an SSL file to work.
*/
/*-------------------------------------------------------------------------*/
/*
   Changes:
  2/1/95replacing all calls to the function Make_DeclarationsNull
    with calls to the function Make_Empty_Declarations

/*-------------------------------------------------------------------------*/
%[
/*-------------------------------------------------------------------------*/
void Enforce_Consistency()
{
  PROD_INSTANCE
  temp_p,
  temp_son,
  IdIsNull();

  void
  Add_New_Ops_To_Proto(),
  Clean_Op(),
  Clean_Data();

#ifdef SDE_DEBUG_1
  printf("Entering Enforce_Consistency\n");
#endif

  Add_New_Ops_To_Proto(),

  temp_p = psdl_components_from_Prot(Global_Proto);

  /* temp_p now points to psdl_components */

  while (production(temp_p) == prod_psdl_pair)
  {
  temp_son = component_from_PsdlPair(temp_p);
  if (production(temp_son) == prod_op)
  {

#ifdef SDE_DEBUG_2
    printf("Enforce_Consistency: Op name = %s\n",
      str0_to_str_ro(StrValue(Get_Op_Name(temp_son))));
```

```
#endif

    if (IntValue(IdIsNull(id_from_Op(temp_son))) == 0)
    {
      Clean_Op(temp_son);
    }
  }
  else if (production(temp_son) == prod_data)
  {
#ifdef SDE_DEBUG_2
    printf("Enforce_Consistency: Type name = %s\n",
      str0_to_str_ro(StrValue(Get_Id(id_from_Data(temp_son)))));
#endif

    if (IntValue(IdIsNull(id_from_Data(temp_son))) == 0)
    {
      Clean_Data(temp_son);
    }
  }


/*
  temp_p = son(temp_p, 2);
*/
  temp_p = psdl_components_from_PsdlPair(temp_p);
  }


#ifdef SDE_DEBUG_1
  printf("Leaving Enforce_Consistency\n");
#endif
}
/*---------------------------------------------------------------------------*/
void Clean_Op(p)
  PROD_INSTANCE p;
{
  HeadPtr
  h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(),
  Make_Operator_Header();

  ATREE
  temptree,
  atree = bu_atree(br_buf(cur_browser));

  PROD_INSTANCE
  temp_p,
  new_op_spec,
  new_op_impl,
  new_inputs_list,
  new_outputs_list,
  new_met,
  new_stream_list,
  new_constraint_list,
  Get_Op_Name(),
  Make_Op(),
  Make_Inputs_List(),
  Make_Outputs_List(),
  Make_New_O_Timing_Info(),
  Build_Stream(),
  Make_StreamsNull(),
  Build_Constraints(),
  Make_ConstraintsNull(),
  Op_Impl_Has_Non_Null_Streams(),
  Op_Impl_Has_Non_Null_Constraints(),
  Replace_Input_Output_Met(),
  Replace_Stream_Constraint_List(),
  Op_Impl_Is_Null(),
  Empty_Graph();
/*
  Make_OpImplNull();
*/

  void
  Print_Type_Decl();

  boolean
  need_new_op_spec,
  need_new_op_impl;

#ifdef SDE_DEBUG_1
  printf("Entering Clean_Op\n");
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);
  if (h == NULL)
  {

#ifdef SDE_DEBUG_2
        /*debugging*/
        printf("Clean_Op: cannot find header_node\n");
#endif

                /* need to make new header node */
        h = Make_Operator_Header(str0_to_str_ro(StrValue(Get_Op_Name(p))
),
         NULL, NULL, Get_Unique_Id(), (int)p, FALSE);
        Link_To_Structure(h);
  }

#ifdef SDE_DEBUG_2
  /* debugging */
  printf("Clean_Op: Op_Name = %s\n", h->name);
#endif

  if (!(h->multi_op_error))
  {
  if (h->input_error)
  {
    /*
    Print_Type_Decl(h->inh_input_decl);
    */

    new_inputs_list = Make_Inputs_List(h->inh_input_decl);
  }
  else
  {
/*
    new_inputs_list = son(son(p, 2), 2);
*/
    new_inputs_list = o_inputs_list_from_OperatorSpec(operator_spec_from_Op(p));
  }

  if (h->output_error)
  {
    /*
```

```
    Print_Type_Decl(h->inh_output_decl);
    */

    new_outputs_list = Make_Outputs_List(h->inh_output_decl);

  }
  else
  {
/*
    new_outputs_list = son(son(p, 2), 3);
*/
    new_outputs_list = o_outputs_list_from_OperatorSpec(operator_spec_from_Op(p));
  }

  if (h->met_error)
  {

#ifdef SDE_DEBUG_2
    /*debugging*/
    if (production(o_timing_info_from_OperatorSpec(
      operator_spec_from_Op(p))) == op_search("OpTimingInfo"))
    {

      printf("Clean_Op: old_met = %d\n", IntValue(
          Convert_Time_To_Integer(
            time_from_OpTimingInfo(
              o_timing_info_from_OperatorSpec(
                operator_spec_from_Op(p))))));
    }
    else
    {
      printf("Clean_Op: old_met = 0\n");
    }

    printf("Clean_Op: new_met = %d\n", IntValue(
          Convert_Time_To_Integer(h->inh_met)));
#endif

    new_met = Make_New_O_Timing_Info(h->inh_met,
        o_timing_info_from_OperatorSpec(operator_spec_from_Op(p)));
  }
  else
  {
    new_met = o_timing_info_from_OperatorSpec(operator_spec_from_Op(p));
  }

  if (h->input_error || h->output_error || h->met_error)
  {
    h->input_error = false;
    h->output_error = false;
    h->met_error = false;

/*
    temp_p = son(p, 2);
*/
    temp_p = operator_spec_from_Op(p);
    new_op_spec = Replace_Input_Output_Met(temp_p, new_inputs_list,
            new_outputs_list, new_met);
    need_new_op_spec = true;
  }
  else
  {
    new_op_spec = operator_spec_from_Op(p);
    need_new_op_spec = false;
  }

/*
  if ((IntValue(Get_Impl_Form(operator_impl_from_Op(p))) > 0)
    && (BoolValue(Empty_Graph(operator_impl_from_Op(p)))))
  {
    need_new_op_impl = true;
    new_op_impl = Make_OpImplNull();

  }
  else
  {
*/

    if (h->stream_error)
    {
      new_stream_list = Build_Streams(
          IdSetDifference(
            h->edge_id_set,
            IdSetUnion(
                h->state_id_set,
                IdSetUnion(
                    h->inh_input_id_set,
                    h->inh_output_id_set))),

          Global_Type_Decl);
    }
    else
    {

      if (BoolValue(Op_Impl_Has_Non_Null_Streams(operator_impl_from_Op(p))))
      {
        new_stream_list= optional_streams_from_Declarations(
            declarations_from_OperatorImpl(
                operator_impl_from_Op(p)));
      }
      else
        new_stream_list = Make_StreamsNull();
    }

    if (h->constraint_error)
    {

      if (BoolValue(Op_Impl_Has_Non_Null_Constraints(operator_impl_from_Op(p))))
      {
        new_constraint_list = Build_Constraints(h->vertex_id_set,
            constraints_from_Cc(
              cc_from_OperatorImpl(
                operator_impl_from_Op(p))));
      }
      else
        new_constraint_list = Build_Constraints(
          h->vertex_id_set, Make_ConstraintsNull());
    }
    else
    {

      if (BoolValue(Op_Impl_Has_Non_Null_Constraints(operator_impl_from_Op(p))))
      {
        new_constraint_list = constraints_from_Cc(
            cc_from_OperatorImpl(
              operator_impl_from_Op(p)));
```

```
      }
      else
        new_constraint_list = Make_ConstraintsNull();
    }


    if (h->stream_error || h->constraint_error)
    {
      h->stream_error = false;
      h->constraint_error = false;

      temp_p = operator_impl_from_Op(p);
      new_op_impl = Replace_Stream_Constraint_List(temp_p,
            new_stream_list, new_constraint_list);
      need_new_op_impl = true;
    }
    else
    {
      new_op_impl = operator_impl_from_Op(p);
      need_new_op_impl = false;
    }
/*
  }
*/

  if (need_new_op_spec || need_new_op_impl)
  {
    need_new_op_spec = false;
    need_new_op_impl = false;

    temptree= tree_to_atree(Make_Op(id_from_Op(p),
             new_op_spec, new_op_impl));
    atree_is_not_maintained(temptree) = true;

        insert_placeholder_and_set_selection(atree, p);

        if (context(temptree) != context(atree))
        {
                insert_coersion(atree, temptree);
        }
        else
        {
                swap_selections(temptree, atree);
        }

        establish_resting_place(atree);
        rm_atree(temptree);
  }

  }
}


/*-------------------------------------------------------------------------*/
void Clean_Data(p)
  PROD_INSTANCE p;
{
  HeadPtr
  h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(),
  Make_Operator_Header();

  TYPE_LIST
  tl,
  Make_Type_Node(),
  Find_Type_Node_from_Data();

  ATREE
  temptree,
  atree = bu_atree(br_buf(cur_browser));

  PROD_INSTANCE
  temp_p,
  temp_type_impl,
  temp_op_impl_list,
  temp_t_op_impl,
  new_op_impl,
  new_stream_list,
  new_constraint_list,
  Make_TOpImpl(),
  Make_TOpImplNull(),
  Build_Stream(),
  Make_StreamsNull(),
  Build_Constraints(),
  Make_ConstraintsNull(),
  Op_Impl_Has_Non_Null_Streams(),
  Op_Impl_Has_Non_Null_Constraints(),
  Replace_Stream_Constraint_List(),
  Op_Impl_Is_Null(),
  Make_New_Op_Impl_List(),
  Concat_Op_Impl_List(),
  Make_TypeImpl();
/*
  Make_OpImplNull();
*/

  void
  Print_Type_Decl();

  boolean
  need_new_op_impl;

#ifdef SDE_DEBUG_1
  printf("Entering Clean_Data\n");
#endif


  tl = Find_Type_Node_from_Data(p);
  if (tl == NULL)
  {
  tl = Make_Type_Node(str0_to_str_ro(StrValue(id_from_Data(p))));
  }

  temp_type_impl = type_impl_from_Data(p);
  if (production(temp_type_impl) == prod_type_impl)
  {

  temp_op_impl_list =
    operator_impl_list_from_TypeImpl(temp_type_impl);

  while (production(temp_op_impl_list) == prod_op_impl_list_pair)
  {


    temp_t_op_impl = t_oper_impl_from_OpImplListPair(
                temp_op_impl_list);
```

```
    if (production(temp_t_op_impl) == prod_t_op_impl)
    {
      h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(temp_t_op_impl);
      if (h == NULL)
      {

#ifdef SDE_DEBUG_3
        /*debugging*/
        printf("Clean_Data: cannot find header_node\n");
#endif

                /* need to make new header node */
        h = Make_Operator_Header(
           str0_to_str_ro( StrValue(Get_Id(
             id_from_TOpImpl(temp_t_op_impl)))),
           NULL, NULL, Get_Unique_Id(), (int)p, FALSE);
        Link_To_Structure(h);
      }

#ifdef SDE_DEBUG_3
      /* debugging */
      printf("Clean_Data: Op_Name = %s\n", h->name);
#endif

      if (BoolValue(IsElement(id_from_TOpImpl(temp_t_op_impl), tl->obsolete_op_impl)))
      {
        temptree= tree_to_atree(Make_TOpImplNull());
        atree_is_not_maintained(temptree) = true;

        insert_placeholder_and_set_selection(atree, temp_t_op_impl);

        if (context(temptree) != context(atree))
        {
                insert_coersion(atree, temptree);
        }
        else
        {
                swap_selections(temptree, atree);
        }

        establish_resting_place(atree);
        rm_atree(temptree);
      }
      else if (!(h->multi_op_error))
      {
/*
        if (BoolValue(Empty_Graph(
          operator_impl_from_TOpImpl(temp_t_op_impl))))
        {
          need_new_op_impl = true;
               new_op_impl = Make_OpImplNull();
        }
        else
        {
*/
          if (h->stream_error)
          {
            new_stream_list = Build_Streams(
                    IdSetDifference(
                    h->edge_id_set,
                    IdSetUnion(
                      h->state_id_set,
                      IdSetUnion(
                        h->inh_input_id_set,
                        h->inh_output_id_set))),

                Global_Type_Decl);
          }
          else
          {

          if (BoolValue(Op_Impl_Has_Non_Null_Streams(
                operator_impl_from_TOpImpl(temp_t_op_impl))))
            {
                new_stream_list= optional_streams_from_Declarations(
                    declarations_from_OperatorImpl(
                    operator_impl_from_TOpImpl(temp_t_op_impl)));
            }
            else
                new_stream_list = Make_StreamsNull();
          }

          if (h->constraint_error)
          {
            if (BoolValue(Op_Impl_Has_Non_Null_Constraints(
                operator_impl_from_TOpImpl(temp_t_op_impl))))
            {
                new_constraint_list = Build_Constraints(h->vertex_id_set,
                        constraints_from_Cc(
                        cc_from_OperatorImpl(
                        operator_impl_from_TOpImpl(temp_t_op_impl))));
            }
            else
            {
                new_constraint_list = Build_Constraints(
                    h->vertex_id_set, Make_ConstraintsNull());
            }
          }
          else
          {
            if (BoolValue(Op_Impl_Has_Non_Null_Constraints(
                operator_impl_from_TOpImpl(temp_t_op_impl))))
            {
                new_constraint_list = constraints_from_Cc(
                        cc_from_OperatorImpl(
                        operator_impl_from_TOpImpl(temp_t_op_impl)));
            }
            else
            {
                new_constraint_list = Make_ConstraintsNull();
            }
          }

          if (h->stream_error || h->constraint_error)
          {
            h->stream_error = false;
            h->constraint_error = false;

            temp_p = operator_impl_from_TOpImpl(temp_t_op_impl);
            new_op_impl = Replace_Stream_Constraint_List(temp_p,
                    new_stream_list, new_constraint_list);
            need_new_op_impl = true;
          }
          else
          {
```

```
            new_op_impl = operator_impl_from_TOpImpl(temp_t_op_impl);
            need_new_op_impl = false;
          }
/*
        }
*/

        if (need_new_op_impl)
        {
          need_new_op_impl = false;

          temptree= tree_to_atree(Make_TOpImpl(
                id_from_TOpImpl(temp_t_op_impl),
                new_op_impl));
          atree_is_not_maintained(temptree) = true;

        insert_placeholder_and_set_selection(atree, temp_t_op_impl);

        if (context(temptree) != context(atree))
        {
                insert_coersion(atree, temptree);
        }
        else
        {
                swap_selections(temptree, atree);
        }

        establish_resting_place(atree);
        rm_atree(temptree);
        }

      }
    }

    temp_op_impl_list = operator_impl_from_TOpImpl(
            temp_op_impl_list);
  }
  }

  /* add the missing op impl */
  if (!BoolValue(IsNull(tl->undefined_op_impl)))
  {
  if (production(temp_type_impl) == prod_type_impl)
  {
    temp_op_impl_list =
                operator_impl_list_from_TypeImpl(temp_type_impl);

    temptree = tree_to_atree(Concat_Op_Impl_List(
            Make_New_Op_Impl_List(tl->undefined_op_impl),
            temp_op_impl_list));

                atree_is_not_maintained(temptree) = true;

                insert_placeholder_and_set_selection(atree, temp_op_impl_list);

                if (context(temptree) != context(atree))
                {
                insert_coersion(atree, temptree);
                }
                else
                {
                swap_selections(temptree, atree);
                }

                establish_resting_place(atree);
                rm_atree(temptree);
        }
/*
  else
  {
    temptree = tree_to_atree(Make_TypeImpl(
        Make_New_Op_Impl_List(tl->undefined_op_impl)));
                atree_is_not_maintained(temptree) = true;

                insert_placeholder_and_set_selection(atree, type_impl_from_Data(p));

                if (context(temptree) != context(atree))
                {
                insert_coersion(atree, temptree);
                }
                else
                {
                swap_selections(temptree, atree);
                }

                establish_resting_place(atree);
                rm_atree(temptree);
        }
*/
  }
}


/*-------------------------------------------------------------------------*/
void Add_New_Ops_To_Proto()
{
  PROD_INSTANCE
  IsNull(),
  Make_Proto(),
  Merge_Psdl_Components(),
  temp_op_set,
  temp_p,
  temp_new_p,
  component_p,
  Get_Op_Name(),
  Make_New_Ops();

  ATREE
        temptree,
        atree = bu_atree(br_buf(cur_browser));

  char
  *component_name;

  boolean
  not_found;

#ifdef SDE_DEBUG_1
  printf("Entering Add_New_Ops_To_Proto\n");
#endif

  temp_op_set = Global_Undef_Ops;
  if (!BoolValue(IsNull(temp_op_set)))
  {
  temp_new_p = Merge_Psdl_Components(
      Make_New_Ops(temp_op_set), psdl_components_from_Prot(Global_Proto));
```

```
/*
      Make_New_Ops(temp_op_set), son(Global_Proto, 1));
*/

  move_selection(atree, one_point_selection(Global_Proto));
  temptree = tree_to_atree(Make_Proto(temp_new_p));
  atree_is_not_maintained(temptree);

       if (context(temptree) != context(atree))
  {
#ifdef SDE_DEBUG_3
  printf("Add_New_Ops: insert_coersion\n");
#endif
                insert_coersion(atree, temptree);
  }
       else
  {
#ifdef SDE_DEBUG_3
    printf("Add_New_Ops: swap_selection\n");
#endif
                swap_selections(temptree, atree);
  }
       establish_resting_place(atree);
       rm_atree(temptree);
       br_set_insert_pt_to_selection(cur_browser);
       cmd_cond_modifies(cur_browser, cur_buffer);
       br_paint_all();

   }
}

/*-------------------------------------------------------------------------*/
FOREIGN Build_InputsList(i, o, td)
  PROD_INSTANCE
  i,/* id */
  o,/* psdl_components consisting of all operators */
  td;/* all type declarations */
{
  PROD_INSTANCE
  temp_input_ids,
  temp_type_decl,
  Extract_Input_Id_Set(),
  IsNull(),
  Make_InputsListPair_From_Type_Decl(),
  Make_InputsListNone();



  temp_input_ids = Extract_Input_Id_Set(i, o);

  if (!BoolValue(IsNull(temp_input_ids)))
  {
  temp_type_decl = Build_Type_Decl(temp_input_ids, td);
  return(Make_InputsListPair_From_Type_Decl(temp_type_decl));
  }
  else
  {
  return(Make_InputsListNone());
  }
}

/*-------------------------------------------------------------------------*/
FOREIGN Build_OutputsList(i, o, td)
  PROD_INSTANCE
  i,/* id */
  o,/* psdl_components consisting of all operators */
  td;/* all type declarations */
{
  PROD_INSTANCE
  temp_input_ids,
  temp_type_decl,
  Extract_Output_Id_Set(),
  IsNull(),
  Make_OutputsListPair_From_Type_Decl(),
  Make_OutputsListNone();



  temp_input_ids = Extract_Output_Id_Set(i, o);

  if (!BoolValue(IsNull(temp_input_ids)))
  {
  temp_type_decl = Build_Type_Decl(temp_input_ids, td);
  return(Make_OutputsListPair_From_Type_Decl(temp_type_decl));
  }
  else
  {
  return(Make_OutputsListNone());
  }
}

/*-------------------------------------------------------------------------*/
void Free_Linked_List(l_l)
  LINKED_LIST l_l;
{
  LINKED_LIST head;

  while (l_l != NULL)
  {
       head = l_l;
       l_l = l_l->next;
       free(head);
  }

}

/*-------------------------------------------------------------------------*/
void Print_Type_Decl(td)
  PROD_INSTANCE td;
{

  PROD_INSTANCE
  temp_p,
  Get_Id(),
  Get_Id_From_Type_Decl();

  temp_p = td;
  while (production(temp_p) == prod_type_decl)
  {
    printf("Print_Type_Decl: Id = %s\n",
        str0_to_str_ro(StrValue(Get_Id(Get_Id_From_Type_Decl(temp_p)))));

    temp_p = type_declarations_from_TypeDeclPair(temp_p);
  }
}
/*-------------------------------------------------------------------------*/
```

```
void Add_Root_Op(name)
  char *name;
{
  ATREE atree = bu_atree(br_buf(cur_browser));
  ATREE temptree;

  PROD_INSTANCE
  Make_Op_From_SSLstring(),
  temp_p,
  temp_new_p;

  PRODUCTION
  p_comp_prod = op_search("PsdlPair");


#ifdef SDE_DEBUG_1
  printf("Entering Add_Root_Op\n");
#endif


  temp_p = psdl_components_from_Prot(Global_Proto);


  insert_placeholder_and_set_selection(atree, temp_p);

  temp_new_p = Make_Op_From_SSLstring(SSLstring(name));
  temptree = tree_to_atree(temp_new_p);

  atree_is_not_maintained(temptree) = true;


        if (context(temptree) != context(atree))
        {
                insert_coersion(atree, temptree);
        }
        else
        {
                swap_selections(temptree, atree);
        }

        establish_resting_place(atree);
        rm_atree(temptree);

        /* update atree, buffer and selection */
        br_set_insert_pt_to_selection(cur_browser);
        cmd_cond_modifies(cur_browser, cur_buffer);

        br_paint_all();

#ifdef SDE_DEBUG_1
  printf("Leaving Add_Root_Op\n");
#endif

}
/*-----------------------------------------------------------------------------*/
void sort_psdl_components()
{
  ATREE
  temptree,
  atree = bu_atree(br_buf(cur_browser));

  PROD_INSTANCE
  p,
  top_id,
  temp_p,
  Sort_Psdl_Compoonents(),
  Make_IdNull(),
  Make_Proto();

  LINKED_LIST
        temp_head,
        current_pos_trace = NULL;

  void
        Free_Linked_List();

#ifdef SDE_DEBUG_1
  printf("Entering sort_psdl_components\n");
#endif

  /* remember the cursor position */
  p = selection_apex(SE_selection(bu_atree(br_buf(cur_browser))));

  temp_p = p;
  current_pos_trace = NULL;

  while (!at_top(temp_p) && (production(temp_p) != prod_op)
                                && (production(temp_p) != prod_data))
  {

#ifdef SDE_DEBUG_3
        /*debugging*/
        printf("sort_psdl_components: son_number = %d\n", son_number(temp_p));
#endif

        temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
        temp_head->item_number = son_number(temp_p);
        temp_head->next = current_pos_trace;
        current_pos_trace = temp_head;

        temp_p = father(temp_p);
  }

  if (!at_top(temp_p) && (production(temp_p) != prod_no_component))
  {
  top_id = id_from_Op_or_Data(temp_p);
  }
  else
  {
  top_id = Make_IdNull();
  }

#ifdef SDE_DEBUG_3
  printf("sort_psdl_components: top_id = %s\n",
  str0_to_str_ro(StrValue(Get_Id(top_id))));
#endif

  move_selection(atree, one_point_selection(Global_Proto));


  temptree = tree_to_atree(Make_Proto(Sort_Psdl_Components(
          psdl_components_from_Prot(Global_Proto))));

  atree_is_not_maintained(temptree);
```

```
  if (context(temptree) != context(atree))
        insert_coersion(atree, temptree);
  else
        swap_selections(temptree, atree);

  establish_resting_place(atree);
  rm_atree(temptree);
  br_set_insert_pt_to_selection(cur_browser);
  cmd_cond_modifies(cur_browser, cur_buffer);
  br_paint_all();

#ifdef SDE_DEBUG_3
  printf("sort_psdl_components: finished swapping trees\n");
#endif

  /* move cursor back to original position */
  if (IntValue(IdIsNull(top_id)) == 0)
  {
  temp_p = Find_Component(top_id, psdl_components_from_Prot(Global_Proto));
  temp_head = current_pos_trace;
  while (temp_head != NULL)
  {
    temp_p = son(temp_p, (temp_head->item_number));
    temp_head = temp_head->next;

  }
  p = temp_p;
  Free_Linked_List(current_pos_trace);

        temp_p = one_point_selection(p);
        move_selection(atree, temp_p);
        br_set_insert_pt_to_selection(cur_browser);
        cmd_cond_modifies(cur_browser, cur_buffer);

        br_paint_all();

  }

#ifdef SDE_DEBUG_1
  printf("Leaving sort_psdl_components\n");
#endif



}

/*---------------------------------------------------------------------------*/
void edit_graph()
{
  ATREE
  temptree,
  atree = bu_atree(br_buf(cur_browser));

  PROD_INSTANCE
  junk_p;

  PROD_INSTANCE
  IsElement(),
  new_graph,
  Edit_Graph(),
  Make_Op_impl(),
  Make_CcNull(),
/*
  Make_DeclarationsNull(),
*/
  Make_Empty_Declarations(),
  Make_Id_From_SSLstring(),
  Get_Impl_Form(),
  Find_Component(),
  Find_Op_Impl_In_Data(),
  current_op_id,
  component_id,
  op_impl_p,
  temp_p;

  PRODUCTION
  top_production;

  HeadPtr
        h,
  parent_head_ptr,
  Find_Parent_Name(),
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  char
  *vertex_type_name,
  *component_name;

  void
        House_Cleaning(),
  Refresh_Graph_Viewer();


#ifdef SDE_DEBUG_1
  printf("Entering edit_graph\n");
#endif

  temp_p = selection_apex(SE_selection(bu_atree(br_buf(cur_browser))));

  while (!at_top(temp_p) && (production(temp_p) != prod_op) && (production(temp_p) !=
prod_t_op_impl))
  {
        temp_p = father(temp_p);
  }

  if (at_top(temp_p))
  {
  write_error_string("must select an operator before invoking edit-graph");
        br_paint_all();

  }
  else
  {
  if (production(temp_p) == prod_op)
  {
    current_op_id = id_from_Op(temp_p);
    op_impl_p = operator_impl_from_Op(temp_p);
    top_production = prod_op;
  }
  else
  {
    current_op_id = id_from_TOpImpl(temp_p);
    op_impl_p = operator_impl_from_TOpImpl(temp_p);
    top_production = prod_t_op_impl;
  }
```

```
  if (IntValue(IdIsNull(current_op_id)) != 0)
  {
    write_error_string("need to give operator name before invoking edit-graph");
        br_paint_all();
  }
  else
  {
    Global_Refresh_Graph_Viewer = true;

    /* initialize the global variables error_message and ge_result */
    if (error_message != NULL)
        free(error_message);
    ge_result = NO_UPDATE;

    new_graph = Edit_Graph();

        if (ge_result == GO_UP || ge_result == SAME_LEVEL || ge_result == GO_DOWN)
    {


      if (IntValue(Get_Impl_Form(op_impl_p)) == 0)
      {
#ifdef SDE_DEBUG_3
printf("edit_graph: make_Op_Impl\n");
#endif
        temptree = tree_to_atree(
/*
          Make_Op_Impl(new_graph, Make_DeclarationsNull(),
*/
          Make_Op_Impl(new_graph, Make_Empty_Declarations(),
                Make_CcNull()));
      }
      else
      {
#ifdef SDE_DEBUG_3
printf("edit_graph: make_new_graph\n");
#endif
        temptree = tree_to_atree(
          Make_Op_Impl(new_graph,
                declarations_from_OperatorImpl(op_impl_p),
                cc_from_OperatorImpl(op_impl_p)));
      }

      atree_is_not_maintained(temptree);
      insert_placeholder_and_set_selection(atree, op_impl_p);

        if (context(temptree) != context(atree))
              insert_coersion(atree, temptree);
        else
              swap_selections(temptree, atree);
#ifdef SDE_DEBUG_3
printf("edit_graph: after tree surgery\n");
#endif

        establish_resting_place(atree);
        rm_atree(temptree);
        br_set_insert_pt_to_selection(cur_browser);
        cmd_cond_modifies(cur_browser, cur_buffer);
        br_paint_all();

      House_Cleaning(SE_selection(bu_atree(br_buf(cur_browser))));
```

```
      if (ge_result == GO_UP)
      {
#ifdef SDE_DEBUG_3
printf("edit_graph: GO_UP\n");

printf("edit_graph: current_op_id = %s\n",
str0_to_str_ro(StrValue(Get_Id(current_op_id))));
#endif
        if (BoolValue(IsElement(current_op_id, Global_Roots))
          || (top_production == prod_t_op_impl))
        {
#ifdef SDE_DEBUG_3
printf("edit_graph: current_op is root\n");
#endif
          write_error_string("operator has no parent, can't edit parent");
          component_name = "";
        }
        else
        {
#ifdef SDE_DEBUG_3
printf("edit_graph: current_op is not root\n");
#endif
          temp_p = Find_Component(current_op_id,
                psdl_components_from_Prot(Global_Proto));

#ifdef SDE_DEBUG_3
printf("edit_graph: returned from Find_Component\n");
#endif

          h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(temp_p);

#ifdef SDE_DEBUG_3
printf("edit_graph: returned from Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec\n");
#endif
          if (strcmp(h->parent, "") == 0)
          {
#ifdef SDE_DEBUG_3
printf("edit_graph: h->parent == empty\n");
#endif
            parent_head_ptr = Find_Parent_Name(h->name);

#ifdef SDE_DEBUG_3
if (parent_head_ptr != NULL)
printf("edit_graph: parent_head_ptr->name = %s\n", parent_head_ptr->name);
else
printf("edit_graph: parent_head_ptr->name = NULL\n");
#endif


            h->parent = strdup(parent_head_ptr->name);


            h->parent_type_name = strdup(parent_head_ptr->type_id);

#ifdef SDE_DEBUG_3
printf("edit_graph: h->parent_type_name = %s\n", h->parent_type_name);
#endif
          }

          if (strcmp(h->parent_type_name, "") != 0)
            component_name = h->parent_type_name;
          else
            component_name = h->parent;;
```

```
#ifdef SDE_DEBUG_3
printf("edit_graph: parent_component_name = #%s#\n", component_name);
#endif
        }
      }
      else if (ge_result == GO_DOWN)
      {
        if (strcmp(goto_child, "") == 0)
        {
          write_error_string("must select a labelled operator to be decomposed before
exiting graph editor");
          component_name = "";
        }
        else
        {

          vertex_type_name = get_vertex_type_name(goto_child);
          if (strcmp(vertex_type_name, "") != 0)
          {

            write_error_string("can't decompose type-operator within an operator
impl");
            component_name = "";
          }
          else
          {

            component_name = get_vertex_oper_name(goto_child);
          }
        }

      }
      else
      {
        component_name = "";
      }

      if (strcmp(component_name, "") != 0)
      {
/*
printf("edit_graph: component_name =%s\n", component_name);
*/

/*
        component_id = Make_Id_From_SSLstring(SSLstring(component_name));
*/
        component_id = Id(SSLstring(component_name));
        temp_p = Find_Component(component_id,
            psdl_components_from_Prot(Global_Proto));

        if (production(temp_p) != prod_no_component)
        {
          if (production(temp_p) == prod_data)
          {
            temp_p = type_impl_from_Data(temp_p);
            if (production(temp_p) != prod_type_impl)
            {
                write_error_string("parent has empty type impl");
            }
            else
            {
                temp_p = Find_TOpImpl_in_operator_impl_list(
                            Make_Id_From_SSLstring(
                                SSLstring(h->parent)),
                            operator_impl_list_from_TypeImpl(temp_p));

#ifdef SDE_DEBUG_3
printf("edit_graph: New Op Id = %s\n", str0_to_str_ro(StrValue(Get_Id(
      id_from_TOpImpl(temp_p)))));
#endif

                move_selection(atree, one_point_selection(
                        id_from_TOpImpl(temp_p)));
            }
          }
          else
          {
            move_selection(atree, one_point_selection(id_from_Op(temp_p)));
          }

        br_set_insert_pt_to_selection(cur_browser);
        cmd_cond_modifies(cur_browser, cur_buffer);
        br_paint_all();

/*
      Refresh_Graph_Viewer();
*/
          edit_graph();
        }
      }
      else
      {
        Refresh_Graph_Viewer();
      }

    }
    else if (ge_result == ERROR)
    {
      write_error_string(error_message);
    }
  }

  }

#ifdef SDE_DEBUG_1
  printf("Leaving edit_graph\n");
#endif

}
/*-------------------------------------------------------------------------------*/

HeadPtr Find_Parent_Name(child_name)
  char *child_name;
{

  boolean
  not_found = true;

  HeadPtr
  h = prototype;

  OPNodePTR
  p_list;

/*
```

```
  char
  *parent_name;
*/

#ifdef SDE_DEBUG_1
  printf("Entering Find_Parent_Name\n");
#endif


#ifdef SDE_DEBUG_3
printf("Find_Parent_Name: child_name = %s\n", child_name);
#endif

  while (not_found && h != NULL)
  {
  p_list = h->operator_list;
        while (not_found && p_list != NULL)
  {
#ifdef SDE_DEBUG_3
printf("Find_Parent_Name: target_name = %s\n", p_list->op->name);
#endif
    if (strcmp(child_name, p_list->op->name) == 0)
    {
      not_found = false;
/*
      parent_name = h->name;
*/
    }
    else
    {
      p_list = p_list->next;
    }
  }
  if (not_found)
    h = h->next;
  }

  if (not_found)
  {
#ifdef SDE_DEBUG_3
printf("Find_Parent_Name: parent not found\n");
#endif

  return(NULL);
  }
  else
  {
#ifdef SDE_DEBUG_3
printf("Find_Parent_Name: found parent\n");
printf("Find_Parent_Name: h->name = %s\n", h->name);
#endif

  return(h);
  }
#ifdef SDE_DEBUG_1
  printf("Leaving Find_Parent_Name\n");
#endif

}
/*-------------------------------------------------------------------------*/
FOREIGN Get_Ada_Op_Impl()
{

  PROD_INSTANCE
  temp_p,
  Make_AdaOpImpl();


  temp_p = selection_apex(SE_selection(bu_atree(br_buf(cur_browser))));

  while (!at_top(temp_p) && (production(temp_p) != prod_op) && (production(temp_p) !=
prod_t_op_impl))
  {
        temp_p = father(temp_p);
  }

  if (at_top(temp_p))
  {
  write_error_string("must select an operator before invoking transformation");
        br_paint_all();
  return(NULLVALUE);

  }
  else
  {
  if (production(temp_p) == prod_op)
    return(Make_AdaOpImpl(id_from_Op(temp_p)));
  else
    return(Make_AdaOpImpl(id_from_TOpImpl(temp_p)));
  }
}
/*-------------------------------------------------------------------------*/
FOREIGN Remove_Leading_Blanks_From_String(s)
  PROD_INSTANCE(s);
{
  char
  *temp_string;
  int
  i;

  temp_string = str0_to_str_ro(StrValue(s));

#ifdef SDE_DEBUG_3
/*debugging*/
printf("Remove_Leading_Blanks_From_String: temp_string = %s\n", temp_string);
#endif

  while (((*temp_string) != NULL) &&
      (((*temp_string) == ' ') || ((*temp_string) == '')))
    temp_string ++;

  return(CommentLine(SSLstring(temp_string)));
}


/*-------------------------------------------------------------------------*/
FOREIGN Get_Ada_Type_Impl()
{

  PROD_INSTANCE
  temp_p,
  Make_AdaTypeImpl();


  temp_p = selection_apex(SE_selection(bu_atree(br_buf(cur_browser))));
```

```
  while (!at_top(temp_p) && (production(temp_p) != prod_data))
  {
        temp_p = father(temp_p);
  }

  if (at_top(temp_p))
  {
  write_error_string("must select an operator before invoking transformation");
        br_paint_all();
  return(NULLVALUE);
  }
  else
  {
/*
  return(Make_AdaTypeImpl(son(temp_p, 1)));
*/
  return(Make_AdaTypeImpl(id_from_Data(temp_p)));
  }
}

/*-------------------------------------------------------------------------*/
FOREIGN Build_Met(i, o)
  PROD_INSTANCE
  i,/* operator_id */
  o;/* psdl_components consisting of all operators */
{
  PROD_INSTANCE
  temp_met,
  Extract_Met(),
  Valid_Met();

  temp_met = Extract_Met(i, o);
  if (BoolValue(Valid_Met(temp_met)))
  {
  return(Make_OpTimingInfo_From_Met(temp_met));
  }
  else
  {
  return(Make_OpTimingInfoNone());
  }
}

/*-------------------------------------------------------------------------*/
void Remove_Empty_Graph()
{
  PROD_INSTANCE
  p,
  Empty_Graph(),
  Make_Op(),
  Make_OpImplNull(),
  temp_op_impl_list,
  temp_t_op_impl,
  temp_p,
  temp_son;

  ATREE
  temptree,
  atree = bu_atree(br_buf(cur_browser));


  p = psdl_components_from_Prot(Global_Proto);

  while (production(p) == prod_psdl_pair)
  {
  temp_son = component_from_PsdlPair(p);

  if (production(temp_son) == prod_op)
  {
    temp_p = operator_impl_from_Op(temp_son);

    if ((production(temp_p) == prod_op_impl) && BoolValue(Empty_Graph(temp_p)))
    {
      temptree = tree_to_atree(Make_Op(id_from_Op(temp_son),
            operator_spec_from_Op(temp_son),
            Make_OpImplNull()));
      atree_is_not_maintained(temptree) = true;
      insert_placeholder_and_set_selection(atree, temp_son);
      if (context(temptree) != context(atree))
      {
        insert_coersion(atree, temp_son);
      }
      else
      {
        swap_selections(temptree, atree);
      }
      establish_resting_place(atree);
      rm_atree(temptree);
    }

  }
  else
  {
    if (production(type_impl_from_Data(temp_son)) == prod_type_impl)
    {
      temp_op_impl_list = operator_impl_list_from_TypeImpl(
            type_impl_from_Data(temp_son));
        while (production(temp_op_impl_list) == prod_op_impl_list_pair)
          {

                temp_t_op_impl = t_oper_impl_from_OpImplListPair(
                                                      temp_op_impl_list);

                if (production(temp_t_op_impl) == prod_t_op_impl)
                {
          temp_p = operator_impl_from_TOpImpl(temp_t_op_impl);

          if ((production(temp_p) == prod_op_impl)
            && BoolValue(Empty_Graph(temp_p)))
                                {

                                      temptree= tree_to_atree(Make_TOpImpl(
                                                id_from_TOpImpl(temp_t_op_impl),
                                                Make_OpImplNull()));
                                      atree_is_not_maintained(temptree) = true;

                                      insert_placeholder_and_set_selection(atree,temp_t_op_impl);

                                      if (context(temptree) != context(atree))
                                      {
                                              insert_coersion(atree, temptree)
;
                                      }
                                      else
                                      {
                                              swap_selections(temptree, atree)
;
```

```
                                        }

                                        establish_resting_place(atree);
                                        rm_atree(temptree);
                                    }

                                }

                temp_op_impl_list = operator_impl_from_TOpImpl(
                                    temp_op_impl_list);
            }

        }
    }

    p = psdl_components_from_PsdlPair(p);

    }


}
/*-------------------------------------------------------------------------*/
%]


/*************************************************************************
****  This sixth set of functions were already existing and was
****  documented originally as file: ed.10.ssl
*************************************************************************/
/*-------------------- FOREIGN FUNCTION DEFINITIONS -------------------------*/
graph foreign Edit_Graph();
o_inputs_list foreign Build_InputsList(id i, psdl_components o, type_declarations td);
o_outputs_list foreign Build_OutputsList(id i, psdl_components o, type_declarations
td);
o_timing_info foreign Build_Met(operator_id i, psdl_components o);
BOOL foreign Is_Show_Graph_Text_View();
operator_impl foreign Get_Ada_Op_Impl();
type_impl foreign Get_Ada_Type_Impl();
t_oper_spec foreign Find_T_Op_Spec_In_Data(id o_id, id t_id);
commentLine foreign Remove_Leading_Blanks_From_String(CLINE i);

/*-------------------------------------------------------------------------*/
%{#include <sys/types.h>
%}
%{#include <stdlib.h>
%}
%{#include <pwd.h>
%}
%{#include <string.h>
%}
%{#include <malloc.h>
%}
%{#include "/n/sun52/work/mantak/New_Sde/includes.h"
%}
%{#include "/n/sun52/work/mantak/New_Sde/sde_ge.h"
%}
%{#include "/n/sun52/work/mantak/New_Sde/globals.h"
%}
%{#include "/n/sun52/work/mantak/New_Sde/sde_globals.h"
%}
%{#include "/n/sun52/work/mantak/New_Sde/sde_macros.h"
```

```
/*************************************************************************
****  This seventh set of functions were already existing and was
****  documented originally as file: ed.11.ssl
*************************************************************************/
/* The following definitions generate a Graphic Editor entry in the editor's
   main menu.  They are a faithful copy of what is stated in the synthesizer's
   user's manual.  Except that in the manual it is explained that a ".c" file
   with these definitions will do.  While investigating this, I arrived to the
   conclusion that such is not the case.  The definition must be included in
   an SSL file to work.
*/
%{

/*
#define SDE_DEBUG_3 0
#define SDE_DEBUG_2 0
#define GRAPHICS_DEBUG 0
*/

extern BROWSER cur_browser;   /* this is a global pointer to the current
                                 location of the cursor in the editor window
                                 */


extern void refresh();
extern void edit();
extern void kill_viewer();
/*-------------------------------------------------------------------------*/

PROCEDURE call_graphic_editor()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Call_graphic_editor\n");
#endif

  /* the following if statement is added to catch that case
     in which the user invokes the graph editor without
     selecting the structure to be edited.                */

  if (current_graph != NULL) {

    edit();

#ifdef SDE_DEBUG_2
    /* debugging */
    printf("Back from GE\n");
#endif
  }
  else {
    write_error_string("must select an operator before invoking graph editor");
/*
    printf("Error: you need to select the structure to be edited\n");
    printf("       before invoking the graphic editor\n");
*/
  }
}
/*-------------------------------------------------------------------------*/
PROCEDURE save_graphic_attributes()
{
  int
        clean_buffer_no_reclaim();
  extern
```

```
        BUFFER deleted;

  /*  This procedure has two purposes: 1) wipe out the deletion of whatever
          is in the DELETED buffer and 2) write out the C-data structure back to
          a file.
  */

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Save_raphic_attributes\n");
#endif
  clean_buffer_no_reclaim( deleted );
  reclaim();

  /* then save the data structure */
}

/*-------------------------------------------------------------------------*/
FOREIGN Find_T_Op_Spec_In_Data(o_id, t_id)
  PROD_INSTANCE o_id, t_id;
{
  PROD_INSTANCE
  Make_TOpSpecNil(),
  Find_T_Op_Spec(),
  component_id,
  o_operators_p,
  temp_son_p,
  temp_p;

  boolean
  keep_searching = true;

  char
  buffer[100];

  sprintf(buffer, "%s", str0_to_str_ro(StrValue(Get_Id(t_id))));

  temp_p = psdl_components_from_Prot(Global_Proto);
  while (keep_searching)
  {
  if (production(temp_p) == prod_psdl_pair)
  {
    temp_son_p = component_from_PsdlPair(temp_p);
    if (production(temp_son_p) == prod_data)
    {
      component_id = id_from_Data(temp_son_p);
      if (strcmp(str0_to_str_ro(StrValue(Get_Id(component_id))),
            buffer) == 0)
      {
        o_operators_p = o_operators_from_TOpSpec(
                              type_spec_from_Data(temp_son_p));
        return(Find_T_Op_Spec(o_id, o_operators_p));
      }
    }
  }
  else
  {
    keep_searching = false;
  }
  }

  return(Make_TOpSpecNil());
}
```

```
/*-------------------------------------------------------------------------*/
FOREIGN Is_Show_Graph_Text_View()
{
  extern VIEW_NO cur_view;

  return(
    (cur_view == name_to_view("SHOW_GRAPH_TEXT_VIEW"))
    ? SSL_true
    : SSL_false
    );
}
/*-------------------------------------------------------------------------*/
FOREIGN Edit_Graph()
{
  PROD_INSTANCE
  op_id,
  IdIsNull(),
        Make_Graph(),
  old_graph,
  temp_p;
  boolean
  different();
  OPNodePTR
  o_list,
  Sort_Operator_List();
  ST_PTR
  s_list,
  Sort_Stream_List();
  HeadPtr
  h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(),
  Make_Operator_Header();
  void
  Remove_Deleted_Operators(),
  Remove_Deleted_Streams(),
  Restore_Deleted_Operators(),
  Restore_Deleted_Streams(),
  Add_Input_Output_State_Nodes(),
  Remove_Input_Output_State_Nodes(),
  Update_Operator(),
  Move_To_Structure_Front(),
  Link_To_Structure();


  temp_p = selection_apex(SE_selection(bu_atree(br_buf(cur_browser))));

  while (!at_top(temp_p)
    && production(temp_p) != prod_op
    && production(temp_p) != prod_t_op_impl)
  temp_p = father(temp_p);


  if (at_top(temp_p))
  {
  write_error_string("need to select an operator before edit graph");
  return(Make_Graph(NULL, NULL));
  }
  else
  {
  if (production(temp_p) == prod_op)
    op_id = id_from_Op(temp_p);
  else
```

```
    op_id = id_from_TOpImpl(temp_p);

  if (IntValue(IdIsNull(op_id)) != 0)
  {
    write_error_string("need to give operator name before edit graph");
    return(Make_Graph(NULL, NULL));
  }
  else
  {
                h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(temp_p);
                if (h == NULL)
                {

                        /* need to make new header node */
                        h =
Make_Operator_Header(str0_to_str_ro(StrValue(Get_Id(op_id))),
                                                        NULL, NULL, Get_Unique_Id(),
(int)temp_p, FALSE);
                        Link_To_Structure(h);
                }
    else
    {
      Move_To_Structure_Front(h);
    }

    Update_Operator(temp_p, h);
    Add_Input_Output_State_Nodes(h);
    current_graph = h;
    the_operator_list = h->operator_list;
    the_stream_list = h->stream_list;


    /* current_graph cannot be NULL are this point */

#ifdef SDE_DEBUG_3
      printf("------- Edit_Graph: before call_graphic_editor -----------\n");
      Print_OperatorS_Operators(current_graph->operator_list);
      Print_OperatorS_Streams(current_graph->stream_list);
#endif


    call_graphic_editor();


#ifdef SDE_DEBUG_3
      printf("------- Edit_Graph: after call_graphic_editor -----------\n");
      Print_OperatorS_Operators(current_graph->operator_list);
      Print_OperatorS_Streams(current_graph->stream_list);
#endif


    Remove_Input_Output_State_Nodes(h);


    if (ge_result != NO_UPDATE && ge_result != ERROR)
    {

      /* current_graph is a global variable which will be modified by
call_graphic_editor */

      Remove_Deleted_Operators(h);

      Remove_Deleted_Streams(h);

      o_list = Sort_Operator_List(current_graph->operator_list);
      current_graph->operator_list = o_list;

      s_list = Sort_Operator_List(current_graph->stream_list);
      current_graph->stream_list = s_list;

      temp_p = Make_Graph(o_list, s_list);
      return(temp_p);

    }
    else if (ge_result == NO_UPDATE)
    {
      Restore_Deleted_Operators(h);

      Restore_Deleted_Streams(h);
    }
    else
    {
      return(Make_Graph(NULL, NULL));
    }
  }
  }
}
/*-------------------------------------------------------------------------*/

boolean different(p, q)
PROD_INSTANCE p, q;
{
    register int i;
    register PRODUCTION prodp, prodq;
    COMPARISON test;

    if (p == q)
        return(FALSE);
    if (p == NULLVALUE || q == NULLVALUE)
        return(TRUE);

    prodp = production(p);
    prodq = production(q);

    if (prodp != prodq)
      return(TRUE);

    if (GR_atom(prodp) && GR_atom(prodq))
        {
            if (atomic_type(prodp)==atomic_type(prodq))
                return( comparison(atomic_type(prodp),
                                   value_ptr(p),
                                   value_ptr(q)
                                   )
      != Equal
                        );
            else
                return(TRUE);
        }
    else
  {
        for (i = leftmost_son(prodp); i <= rightmost_son(prodp); i++)
            {
           test = different(son(p,i), son(q,i));
            if (test != FALSE)
               return(test);
```

```
          );
        return(FALSE);
  }

}
/*---------------------------------------------------------------------------*/

init_sys3(){
  ME me_user;
  /* extern PROCEDURE call_graphic_editor(); */
  /* extern PROCEDURE save_graphic_attributes(); */

  FILE *tool_file, *fopen();
  char Buffer[100 + 1];

  char home_dir[100];
/*
  struct passwd *user_pass;
*/

  void
  edit_graph(),
  sort_psdl_components(),
  clean_gedatatransfile(),
  enforce_consistency_on(),
  enforce_consistency_off();

  int
  create_sb_query(),
  print_psdl(),
/*
  split_psdl(),
*/
  save_psdl(),
  save_psdl_exit();

  extern HeadPtr
       prototype;

  extern int
  unique_id_count;

  extern PRODUCTION
        prod_prot,
        prod_psdl_pair,
        prod_op,
        prod_data,
        prod_no_component,
        prod_op_spec,
        prod_op_impl,
  prod_type_impl,
        prod_t_op_impl,
        prod_op_impl_list_pair,
        prod_input_list,
        prod_inputs,
        prod_output_list,
        prod_outputs,
  prod_state_list,
        prod_type_decl,
        prod_graph_null,
        prod_graph,
        prod_vertex_list_pair,
  prod_optional_type_id,
        prod_operator_id_pairs,
        prod_alone_id_pair,
        prod_a_decl,
        prod_decl,
        prod_stream,
        prod_cc,
        prod_constraints,
        prod_a_constraint;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering init_sys3\n");
#endif

  /* create new menu */
  me_user = me_open("CAPS-Cmds");

  /* create five menu items */
  (void) co_open(
    "save-psdl",
    save_psdl,
    co_valid
  );

  (void) co_open(
    "save-psdl-exit",
    save_psdl_exit,
    co_valid
  );
#ifdef D_EDITOR
  (void) co_open(
    "enforce-consistency_on",
    enforce_consistency_on,
    co_valid
  );

  (void) co_open(
    "enforce-consistency_off",
    enforce_consistency_off,
    co_valid
  );
#endif

  (void) co_open(
    "sort-psdl-components",
    sort_psdl_components,
    co_valid
  );

  (void) co_open(
    "edit-graph",
    edit_graph,
    co_valid
  );

  (void) co_open(
    "create-sb-query",
    create_sb_query,
    co_valid
  );

  (void) co_open(
    "print-psdl",
```

```
    print_psdl,
    co_valid
  );

/*
  (void) co_open(
    "split-psdl",
    split_psdl,
    co_valid
  );
*/

  if (me_user != ME_NULL)
  {
  MenuItem("edit-graph", me_user);
  MenuItem("create-sb-query", me_user);
  MenuItem("print-psdl", me_user);
/*
  MenuItem("split-psdl", me_user);
*/
  MenuItem("save-psdl", me_user);
  MenuItem("save-as", me_user);
  MenuItem("save-psdl-exit", me_user);
#ifdef D_EDITOR
  MenuItem("enforce-consistency_on", me_user);
  MenuItem("enforce-consistency_off", me_user);
#endif
  MenuItem("sort-psdl-components", me_user);
  MenuItem("exit", me_user);
  }
  else
  {
  printf("init_sys3: error -- cannot find CAPS-Cmds menu\n");
  }

/*
  user_pass = getpwuid(getuid());
  strcpy(home_dir, user_pass->pw_dir);
*/
  strcpy(home_dir, getenv("CAPSHOME"));
  strcat(home_dir, "/bin/tool_location.txt");
  clean_gedatatransfile();

  if((tool_file = fopen(home_dir, "rt")) == NULL)
        {
          printf("Tool location file not found.\n");
        }
  else
        {
          fscanf(tool_file, "%s", Buffer);
          while(strcmp(Buffer, "graph_viewer:") != 0)
                fscanf(tool_file, "%s", Buffer);

          fgets(Buffer, 100, tool_file);
          system(Buffer);
        }
  fclose(tool_file);

  /* initialize the global variables */
        prod_prot = op_search("Prot");
        prod_psdl_nil = op_search("PsdlNil");
        prod_psdl_pair = op_search("PsdlPair");
        prod_op = op_search("Op");
        prod_data = op_search("Data");
        prod_no_component = op_search("NoComponent");
        prod_op_spec = op_search("OperatorSpec");
        prod_op_impl = op_search("OperatorImpl");
        prod_type_impl = op_search("TypeImpl");
        prod_t_op_spec = op_search("TOpSpec");
        prod_t_op_impl = op_search("TOpImpl");
        prod_op_impl_list_pair = op_search("OpImplListPair");
        prod_input_list = op_search("InputsListPair");
        prod_inputs = op_search("OpInputs");
        prod_output_list = op_search("OutputsListPair");
        prod_outputs = op_search("OpOutputs");
  prod_state_list_none = op_search("StatesListNone");
  prod_state_list = op_search("StatesListPair");
        prod_type_decl = op_search("TypeDeclPair");
        prod_graph_null = op_search("GraphNull");
        prod_graph = op_search("Graph");
        prod_vertex_list_pair = op_search("VertexListPair");
  prod_optional_type_id = op_search("OptionalTypeId");
  prod_operator_id_pairs = op_search("OperatorIdPairs");
  prod_alone_id_pair = op_search("AIdPair");
        prod_a_decl = op_search("ADecl");
        prod_decl = op_search("Declarations");
        prod_stream = op_search("Streams");
        prod_cc = op_search("Cc");
        prod_constraints = op_search("ConstraintsPair");
        prod_a_constraint = op_search("AConstraint");

  prototype = NULL;
  Global_Type_List = NULL;

/*
  Global_Operators = NULLVALUE;
*/
  Global_Proto = NULLVALUE;
  Global_Roots = NULLVALUE;
  Global_Type_Decl = NULLVALUE;
  Global_Undef_Ops = NULLVALUE;
  Global_Undef_Type_Ops = NULLVALUE;
  Global_Current_Op_Name = "";
  unique_id_count = 1;
  Global_Refresh_Graph_Viewer = true;

  /* default of editor is to have Enforce_Consistency on */
  Global_Enforce_Consistency = true;

  Global_Set_SdeView = false;
printf("init_sys3: Global_Set_SdeView = false\n");

#ifdef D_EDITOR
  Global_Enforce_Consistency = false;
#endif
}
/*-------------------------------------------------------------------------*/
char * get_prototype_name()
{
  /*This procedure extracts the prototype name from the title of the
  current buffer.
  */

  char
  *path_name,
  *component_name,
```

```
  *last;

  /* attempt to extract a psdl identifier from file name */
  path_name = strdup(bu_file_name(br_buf(cur_browser)));

#ifdef SDE_DEBUG_3
printf("path_name = %s\n", path_name);
#endif

  component_name = strrchr(path_name, '/');
  if (component_name != NULL)
  component_name++; /* skip the `/` character */
  else
  component_name = path_name;

#ifdef SDE_DEBUG_3
printf("component_name = %s\n", component_name);
#endif

  last = component_name;

  if (((*last)>64 && (*last)<91) || ((*last)>96 && (*last)<123))
  {

  last++;
  while ( ((*last)>64 && (*last)<91) ||
       ((*last)>96 && (*last)<123)||
       ((*last)>47 && (*last)<58) ||
       ((*last) == 95))
    last++;
  }
  (*last) = '\0';

#ifdef SDE_DEBUG_3
/*debugging*/
printf("get_prototype_name: component_name = %s\n", component_name);
#endif

  if (strcmp(component_name, "") != 0)
  return(component_name);
  else
  return("_default_proto_name");


}
/*---------------------------------------------------------------------------*/
init_sys6()
{
  /*
        This procedure assumes that the "prototype" data structure is not NULL. if
        such is not the case, there is discrepancy between the attributes file and
        the input text file... execution is thus terminated.

  */


  extern int
        Process_File();

  char
  *component_name,
  *get_prototype_name();

  ATREE
  temptree,
  atree = bu_atree(br_buf(cur_browser));

  PROD_INSTANCE
  Get_Id(),
  Sort_Psdl_Components(),
  Make_Proto(),
  temp_p;

  PROD_INSTANCE
  Change_view();

  void
  Add_Root_Op();

  boolean
  keep_searching;


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering init_sys6\n");
#endif

  if( Process_File() != TRUE)
        {
          printf("INPUT FILES ARE CORRUPTED.  PLEASE CHECK\n");
          exit(777);
        }
  else
        printf("INPUT FILES SUCCESSFULLY LOADED\n");

    Change_view(SSLstring("SDE_VIEW"), SSL_true);
    Global_Set_SdeView = true;
printf("init_sys6: Global_Set_SdeView = true\n");

  component_name = strdup(get_prototype_name());


  /* add operator template with name == component_name */
  if ((production(psdl_components_from_Prot(Global_Proto)) != prod_psdl_pair) &&
  (strcmp(component_name, "_default_proto_name") != 0))
  {
#ifdef SDE_DEBUG_3
/*debugging*/
printf("init_sys6: psdl_components_from_Prot(Global_Proto) == PsdlNil\n");
#endif

  Add_Root_Op(component_name);

  temp_p = graph_from_OperatorImpl(
       operator_impl_from_Op(
         component_from_PsdlPair(
           psdl_components_from_Prot(Global_Proto))));
  move_selection(atree, one_point_selection(temp_p));
        br_set_insert_pt_to_selection(cur_browser);
        cmd_cond_modifies(cur_browser, cur_buffer);
        br_paint_all();

  }
  else
  {
```

```
/*
       if ((production(son(son(Global_Proto, 1), 1)) == prod_no_component) &&
*/
   if ((production(component_from_PsdlPair(
    psdl_components_from_Prot(Global_Proto))) == prod_no_component) &&
    (strcmp(component_name, "_default_proto_name") != 0))

  {
#ifdef SDE_DEBUG_3
/*debugging*/
printf("init_sys6: component_from_PsdlPair(psdl_components_from_Prot(Global_Proto)) ==
NoComponent\n");
#endif

    Add_Root_Op(component_name);

    temp_p = graph_from_OperatorImpl(
        operator_impl_from_Op(
          component_from_PsdlPair(
            psdl_components_from_Prot(Global_Proto))));
    move_selection(atree, one_point_selection(temp_p));
               br_set_insert_pt_to_selection(cur_browser);
               cmd_cond_modifies(cur_browser, cur_buffer);
               br_paint_all();

  }
  else
  {
    /* sort the components alphabetically, putting all Types before Ops */
    move_selection(atree, one_point_selection(Global_Proto));
/*
    temp_p = son(Global_Proto, 1);
*/
    temp_p = psdl_components_from_Prot(Global_Proto);
    temp_p = Sort_Psdl_Components(temp_p);
    temptree = tree_to_atree(Make_Proto(temp_p));
    atree_is_not_maintained(temptree);
    if (context(temptree) != context(atree))
      insert_coersion(atree, temptree);
    else
      swap_selections(temptree, atree);
    establish_resting_place(atree);
    rm_atree(temptree);
               br_set_insert_pt_to_selection(cur_browser);
               cmd_cond_modifies(cur_browser, cur_buffer);
    br_paint_all();

    if (strcmp(component_name, "_default_proto_name") != 0)
    {

      /* position cursor at component with name == component_name */
/*
      temp_p = son(Global_Proto, 1);
*/
      temp_p = psdl_components_from_Prot(Global_Proto);
      keep_searching = true;
      while ((production(temp_p) == prod_psdl_pair) && keep_searching)
      {
        if (strcmp(
            component_name,
/*
                                                StrValue(son(son(son(temp_p, 1), 1),
1)))
*/
            str0_to_str_ro(
                StrValue(
                  Get_Id(
                    id_from_Op(
                      component_from_PsdlPair(temp_p)))))
            ) == 0)
        {
            keep_searching = false;
        }
        else
        {
#ifdef SDE_DEBUG_3
            printf("init_sys6: target component name = %s\n",
                str0_to_str_ro(
                  StrValue(
                    Get_Id(
                      id_from_Op(
                        component_from_PsdlPair(temp_p))))));
#endif

            temp_p = psdl_components_from_PsdlPair(temp_p);
        }
      }

      if (!keep_searching)
      {
        temp_p = component_from_PsdlPair(temp_p);
        if (production(temp_p) == prod_op)
        {
            if (IntValue(Get_Impl_Form(
                operator_impl_from_Op(temp_p))) > 0)
                temp_p = graph_from_OperatorImpl(
                                                    operator_impl_from_Op(temp_p));
        }

        move_selection(atree, one_point_selection(temp_p));
               br_set_insert_pt_to_selection(cur_browser);
               cmd_cond_modifies(cur_browser, cur_buffer);
               br_paint_all();


        /* call Refresh_Graph_Viewer to display the graph */
        /* (MTS) 1/12/95                                  */
        Refresh_Graph_Viewer();
      }


    }
    else
    {
      temp_p = component_from_PsdlPair(
            psdl_components_from_Prot(Global_Proto));
               move_selection(atree, one_point_selection(temp_p));
               br_set_insert_pt_to_selection(cur_browser);
               cmd_cond_modifies(cur_browser, cur_buffer);
               br_paint_all();
    }
  }

    }

    /* change the view from DEFAULT to SDE_VIEW */
```

```
/*
    Change_view(SSLstring("SDE_VIEW"), SSL_false);
    Global_Set_SdeView = true;
printf("init_sys6: Global_Set_SdeView = true");
*/

}
/*-------------------------------------------------------------------------*/
int save_psdl()
{
  PROD_INSTANCE
  Empty_Graph(),
  p,
  temp_p,
  Save_as();

  charbuffer[100];

  intresult = 0;

  void
  Remove_Empty_Graph(),
  save_graphic_file(),
  House_Cleaning();

  LINKED_LIST
  temp_head,
        current_pos_trace = NULL;

  void
        Free_Linked_List();

  ATREE
  atree = bu_atree(br_buf(cur_browser));

  char
  *component_name,
  *get_prototype_name();



  /* clean up the prototype before saving it */
  House_Cleaning(SE_selection(bu_atree(br_buf(cur_browser))));


    /* remember the cursor position */
  p = selection_apex(SE_selection(bu_atree(br_buf(cur_browser))));

  temp_p = p;
  current_pos_trace = NULL;

  while(!at_top(temp_p) && (production(temp_p) != prod_op_impl))
  {
        temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
        temp_head->item_number = son_number(temp_p);
        temp_head->next = current_pos_trace;
        current_pos_trace = temp_head;

        temp_p = father(temp_p);
  }

  if (production(temp_p) == prod_op_impl)
  {
        if (BoolValue(Empty_Graph(temp_p)))
        {
                /* need to replace empty operator impl by OpImplNull
                   and reset cursor position to current point */

                Free_Linked_List(current_pos_trace);
    current_pos_trace = NULL;
        }

        while (!at_top(temp_p))
        {
                temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
                temp_head->item_number = son_number(temp_p);
                temp_head->next = current_pos_trace;
                current_pos_trace = temp_head;

                temp_p = father(temp_p);
        }
  }

  Remove_Empty_Graph();

  /* save graphic information */
  save_graphic_file();

  /* get the prototype name */
  component_name = strdup(get_prototype_name());

  /* create the output file name */
  sprintf(buffer, "%s%s", component_name, ".psdl");

  if (production(psdl_components_from_Prot(Global_Proto)) == prod_psdl_nil)
  {
  write_error_string("empty prototype, nothing to be saved");
  }
  else
  {
  if (IntValue(Save_as(
       SSLstring("Text"),
       SSLstring(buffer),
       SSLstring("BASEVIEW")
/*
       SSLstring("OUTPUT_VIEW")
*/
       )
     ) == 1)
   {
     sprintf(buffer, "%s%s%s", "can't write file ", component_name, ".psdl");
         write_error_string(buffer);
     result = 1;
   }

        if (split_psdl() == 1)
        {
     result = 1;
        }
  }

    /* move cursor back to original position */
  temp_p = Global_Proto;

  temp_head = current_pos_trace;
  while (temp_head != NULL)
```

```
  {
                temp_p = son(temp_p, (temp_head->item_number));
                temp_head = temp_head->next;

  }
  p = temp_p;
  Free_Linked_List(current_pos_trace);

  temp_p = one_point_selection(p);
  move_selection(atree, temp_p);
  br_set_insert_pt_to_selection(cur_browser);
  cmd_cond_modifies(cur_browser, cur_buffer);

  br_paint_all();



  return(result);
}

/*-------------------------------------------------------------------------*/
int print_psdl()
{
  PROD_INSTANCE
  Save_as(),
  result;

  char
  buffer[100];

  char
  *component_name,
  *get_prototype_name();

  /* get the prototype name */
  component_name = strdup(get_prototype_name());

  sprintf(buffer, "%s%s", component_name, ".psdl.printfile");


  result = Save_as(
    SSLstring("Text"),
    SSLstring(buffer),
    SSLstring("BASEVIEW")
/*
    SSLstring("OUTPUT_VIEW")
*/
    );

  if (IntValue(result) != 0)
  {
    write_error_string("can't save psdl, print failed");
    br_paint_all();
    return(1);
  }


  sprintf(buffer, "%s%s%s", "lpr ", component_name, ".psdl.printfile");

  if (IntValue(result) != 0)
  {
    write_error_string("can't save psdl, print failed");
    br_paint_all();
    return(1);
  }

  system(buffer);

  return(IntValue(result));
}

/*-------------------------------------------------------------------------*/
int create_sb_query()
{
  PROD_INSTANCE
  Save_selection_to_file(),
  p,
  temp_p;

  ATREE
  atree = bu_atree(br_buf(cur_browser));

  char  buffer[100];

  intprint_psdl_spec(),
  result;

  LINKED_LIST
        temp_head,
        current_pos_trace = NULL;

  void
        Free_Linked_List();


  char
  *component_name,
  *get_prototype_name();

  /* remember the cursor position */
  p = selection_apex(SE_selection(bu_atree(br_buf(cur_browser))));

  temp_p = p;
  current_pos_trace = NULL;

  /* Note: need to delete (production(temp_p) != prod_data) in
     the following boolean expression if type specs are
     not allowed in sb query Shing 8/8/94
  */

  while (!at_top(temp_p) && (production(temp_p) != prod_op)
                         && (production(temp_p) != prod_data))
  {
        temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
        temp_head->item_number = son_number(temp_p);
        temp_head->next = current_pos_trace;
        current_pos_trace = temp_head;

        temp_p = father(temp_p);
  }

  if (at_top(temp_p))
  {
  write_error_string("must select component before invoking create-sb-query");
  Free_Linked_List(current_pos_trace);
  return(1);
```

```
  }
  else
  {

  /* move selection and cursor to select the whole component */
  move_selection(atree, one_point_selection(temp_p));
        br_set_insert_pt_to_selection(cur_browser);
        cmd_cond_modifies(cur_browser, cur_buffer);
        br_paint_all();

  component_name = strdup(get_prototype_name());

  sprintf(buffer, "%s%s", component_name, ".psdl.sbquery");

  result = print_psdl_spec(buffer);

  /* move cursor back to original position */
        temp_head = current_pos_trace;
        while (temp_head != NULL)
        {
                temp_p = son(temp_p, (temp_head->item_number));
                temp_head = temp_head->next;

        }
        p = temp_p;
        Free_Linked_List(current_pos_trace);

        temp_p = one_point_selection(p);
        move_selection(atree, temp_p);
        br_set_insert_pt_to_selection(cur_browser);
        cmd_cond_modifies(cur_browser, cur_buffer);
  br_paint_all();

  return(result);
  }
}

/*-------------------------------------------------------------------------*/
int split_psdl()
{
  PROD_INSTANCE
  Empty_Graph(),
  p,
  temp_p;

  ATREE
  atree = bu_atree(br_buf(cur_browser));

  char
  temp_buf[100],
  buffer[100],
  error_buf[100];

  LINKED_LIST
        temp_head,
        current_pos_trace = NULL;

  void
        Free_Linked_List();

  int
  result = 0;
```

```
char
*component_name,
*get_prototype_name();


component_name = strdup(get_prototype_name());

/* remember the cursor position */
p = selection_apex(SE_selection(bu_atree(br_buf(cur_browser))));

temp_p = p;
current_pos_trace = NULL;

while(!at_top(temp_p) && (production(temp_p) != prod_op_impl))
{
      temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
      temp_head->item_number = son_number(temp_p);
      temp_head->next = current_pos_trace;
      current_pos_trace = temp_head;

      temp_p = father(temp_p);
}

if (production(temp_p) == prod_op_impl)
{
if (BoolValue(Empty_Graph(temp_p)))
{
  /* need to replace empty operator impl by OpImplNull
     and reset cursor position to current point */

  Free_Linked_List(current_pos_trace);
  current_pos_trace = NULL;
      }

while (!at_top(temp_p))
{
      temp_head = (LINKED_LIST) malloc(sizeof(LINKED_LIST_NODE));
      temp_head->item_number = son_number(temp_p);
      temp_head->next = current_pos_trace;
      current_pos_trace = temp_head;

     temp_p = father(temp_p);
}
}


temp_p = psdl_components_from_Prot(Global_Proto);

while (production(temp_p) == prod_psdl_pair)
{
p = component_from_PsdlPair(temp_p);

if ((production(p) == prod_op) || (production(p) == prod_data))
{

      /* move selection and cursor to select the whole component */
      move_selection(atree, one_point_selection(p));
      br_set_insert_pt_to_selection(cur_browser);
      cmd_cond_modifies(cur_browser, cur_buffer);
      br_paint_all();

  if (strcmp(get_prototype_name(),
       str0_to_str_ro(
```

```
        StrValue(
          Get_Id(
            id_from_Op_or_Data(p))))) == 0)
    {
      sprintf(temp_buf, "%s", component_name);
    }
    else
    {
      sprintf(temp_buf, "%s%s%s", component_name, ".",
            str0_to_str_ro(
            StrValue(
              Get_Id(
                  id_from_Op_or_Data(p)))));
    }

    sprintf(buffer, "%s%s", temp_buf, ".spec.psdl");

    if (print_psdl_spec(buffer) != 0)
    {
      result = 1;
      sprintf(error_buf, "%s%s", "can't write file ", buffer);
      write_error_string(error_buf);
    }

    sprintf(buffer, "%s%s", temp_buf, ".imp.psdl");

    if (print_psdl_impl(buffer) != 0)
    {
      result = 1;
      sprintf(error_buf, "%s%s", "can't write file ", buffer);
      write_error_string(error_buf);
    }
  }

  temp_p = psdl_components_from_PsdlPair(temp_p);
  }

    /* move cursor back to original position */
  temp_p = Global_Proto;
  temp_head = current_pos_trace;
  while (temp_head != NULL)
  {
              temp_p = son(temp_p, (temp_head->item_number));
              temp_head = temp_head->next;

  }
  p = temp_p;
  Free_Linked_List(current_pos_trace);

  temp_p = one_point_selection(p);
  move_selection(atree, temp_p);
  br_set_insert_pt_to_selection(cur_browser);
  cmd_cond_modifies(cur_browser, cur_buffer);

  br_paint_all();

  return(result);

}


/*-------------------------------------------------------------------------*/
```

```
int print_psdl_spec(fn)
  char* fn;
{

  PROD_INSTANCE
  result,
  Save_selection_to_file();

  result = Save_selection_to_file(
       SSLstring("Text"),
       SSLstring(fn),
       SSLstring("SPEC_ONLY_VIEW")
                );
        return(IntValue(result));
}
/*-------------------------------------------------------------------------*/
int print_psdl_impl(fn)
  char* fn;
{
  PROD_INSTANCE
  result,
  Save_selection_to_file();

  result = Save_selection_to_file(
       SSLstring("Text"),
       SSLstring(fn),
       SSLstring("IMPL_ONLY_VIEW")
                );
        return(IntValue(result));
}
/*-------------------------------------------------------------------------*/
int save_psdl_exit()
{
  if (save_psdl() == 0)
  finish(strdup(""));
  return(0);
}
/*-------------------------------------------------------------------------*/
void enforce_consistency_on()
{
  extern boolean Global_Enforce_Consistency;

#ifdef SDE_DEBUG_1
  printf("Entering enforce_consistency_on\n");
#endif

  Global_Enforce_Consistency = true;

#ifdef SDE_DEBUG_1
  printf("Leaving enforce_consistency_on\n");
#endif

}
/*-------------------------------------------------------------------------*/
void enforce_consistency_off()
{
  extern boolean Global_Enforce_Consistency;

#ifdef SDE_DEBUG_1
  printf("Entering enforce_consistency_off\n");
#endif

  Global_Enforce_Consistency = false;
```

```
#ifdef SDE_DEBUG_1
  printf("Leaving enforce_consistency_off\n");
#endif

}
/*-------------------------------------------------------------------------*/
static clean_buffer_no_reclaim( bu )
         BUFFER
           bu;
{
  ATREE atree;
  ATREE old_atree;

  PROD_INSTANCE test_subtree();
  PROD_INSTANCE temp;
    SELECTION s;

  int i;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering Clean_buffer_no_reclaim\n");
#endif

          /* test buffer for completing term  */
          if (is_completing_term(bu_atree(bu)->root)) return;

  /* Put completing production in bu and rm_atree that was there */
  top(bu_atree(bu));

/*
  atree = completing_atree(bu_mode(bu));
  prepareBufferToClear(bu);
  prepareToSwap(bu_atree(bu));
  bu_set_atree(bu, atree);
  restore_tree_after_selection_modification(atree);
*/

/*
  atree = mk_atree(tree_to_abody(test_subtree()));
*/

  old_atree = bu_atree(bu);
  atree = mk_atree(tree_to_abody_reuse(old_atree->root));
  prepareToSwap(old_atree);
  bu_set_atree(bu, atree);
  restore_tree_after_selection_modification(atree);

  temp = selection_apex(SE_selection(atree));

  temp = one_step_forward(temp);

  for (i=1;i<5;i++)
  {
      do {
        temp = one_step_forward_without_descent(temp);
        } while ( (is_a_list_nil(temp) || is_an_opt_empty(temp) ||
                   InTextBuffer(at_text_buffer_table(atree),cur_view,temp))
                && !at_top(temp)
                 );
  }
    if (at_top(temp)) temp = selection_apex(SE_selection(atree));
    remove_extra_placeholders(atree, &temp);
    s = one_point_selection(temp);
    move_selection(atree, s);
    br_set_insert_pt_to_selection(cur_browser);
    cmd_cond_modifies(cur_browser, cur_buffer);

    br_paint_all();

/*
  rm_atree(old_atree);
*/

}
/*-------------------------------------------------------------------------*/
PROCEDURE prepareBufferToClear(buf)
         BUFFER buf;
{
  extern BROWSER
        br_list_head;

  BROWSER
        b;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering prepareBufferToClear\n");
#endif

  for (b = br_list_head; b != BR_NULL; b = br_next(b)) {
        if ( br_buf(b) == buf ) {
          prepareToClear(b);
        }
  }
  return(TRUE);
}
/*-------------------------------------------------------------------------*/

void Link_Stream(p, h)
         ST_PTR p;
         HeadPtr h;
{
#ifdef GRAPHICS_DEBUG
  /* debugging */
  printf("Entering Link_Stream\n");
#endif

  p->next = h->stream_list;
  h->stream_list = p;
}
/*-------------------------------------------------------------------------*/
void Link_Operator(p, h)
         OPNodePTR p;
         HeadPtr h;
{
  /* This procedure takes as input two pointers: "p" and "h".  "p" points to a
         vertex head node which is about to be inserted in the vertex list pointed
         to by "h->operator_list".
  */

#ifdef GRAPHICS_DEBUG
  /* debugging */
  printf("Entering Link_Operator\n");
```

```
#endif

  p->next = h->operator_list;
  h->operator_list = p;
}
/*-------------------------------------------------------------------------*/
#ifdef CANCEL_TEMP_DELETE
mark_killed_operators()
{
  /* this traverses the "deleted" buffer looking for operators that be there.
        For each operator found in the buffer, the "prototype" list is searched
        for based on the pointer to that operator.  The "marked_for_delete" field
        of this operator is then turned to TRUE.
  */

  ATREE
        the_tree;

  PROD_INSTANCE
        root_of_tree,
        component,
        p;

  PRODUCTION
        psdl_pair,
        psdl_nil,
        op;

  int
        key;

  HeadPtr
        h,
        Mark_Operator();

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("operators external computer entered\n");
  printf("Entering mark_killed_operators\n");
#endif


  /* get access to the "deleted" buffer's abstract tree */
  the_tree = bu_atree( deleted );

#ifdef SDE_DEBUG_2
  printf("got deleted tree\n");
#endif

  psdl_pair = op_search("PsdlPair");
  psdl_nil = op_search("PsdlNil");
  op = op_search("Op");


  /* get access to the top_node*/
  p = the_tree->root;

#ifdef SDE_DEBUG_2
  printf("got root node\n");
#endif

  if (p == NULLVALUE) return;

  if (production(p) == op)
        {
          key = (int)p;

#ifdef SDE_DEBUG_2
          printf("call mark with-->%d\n", key);
#endif

          h = Mark_Operator(key);
        }
  else
        {
          if (production(p) == psdl_pair)
                {
                  while (production(p) != psdl_nil)
                        {

#ifdef SDE_DEBUG_2
                          printf("list of components detected\n");
#endif

                          /* Get the non-list element of p */
/*
                          component = son(p, 1);
*/
                          component = component_from_PsdlPair(p);
                          key = (int)component;

#ifdef SDE_DEBUG_2
                          printf("call mark with-->%d\n", key);
#endif
                          h = Mark_Operator(key);
/*
                          p = son(p, 2);
*/
                          p = psdl_components_from_PsdlPair(p);
                        }
                }
            else
                {
                  /* we are not interested in any other type of production */
                }
        }
}
#endif /* CANCEL_TEMP_DELETE */
/*-------------------------------------------------------------------------*/
HeadPtr Mark_Operator(key)
         int
           key;
{
  HeadPtr
        h,
        Op_Number_Is_In_List();

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering mark_operator\n");
#endif

#ifdef SDE_DEBUG_2
  printf("MARKING OPERATOR--> %d\n", key);
#endif
```

```
  h = Op_Number_Is_In_List(key);
  if (h != NULL)
        {
#ifdef SDE_DEBUG_2
          printf("OPERATOR MARKED-->%s\n", h->name);
#endif

          h->marked_for_delete = TRUE;
        }
  else
        {
          printf("NO OPERATOR MARKED.  NUMBER IS NULL\n");
        }
  return(h);
}
/*-------------------------------------------------------------------------*/

global_proto_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_proto_store_init\n");
  printf("Leaving global_proto_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE global_proto_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_proto_store_fetch\n");
  printf("Leaving global_proto_store_fetch\n");
#endif
}
/*-------------------------------------------------------------------------*/
global_proto_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_proto_store_delete\n");
  printf("Leaving global_proto_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
global_proto_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_proto_store_insert\n");
#endif

  Global_Proto = v;  /* Global_Proto is global variable */

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("leaving global_proto_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/

global_root_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_root_store_init\n");
  printf("Leaving global_root_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE global_root_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_root_store_fetch\n");
  printf("Leaving global_root_store_fetch\n");
#endif
}
/*-------------------------------------------------------------------------*/
global_root_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_root_store_delete\n");
  printf("Leaving global_root_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
global_root_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_root_store_insert\n");
#endif

  Global_Roots = v;  /* Global_Roots is global variable */

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("leaving global_root_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
#ifdef CANCEL_TEMP_DELETE

global_operators_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_operators_store_init\n");
  printf("Leaving global_operators_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE global_operators_store_fetch()
```

```
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_operators_store_fetch\n");
  printf("Leaving global_operators_store_fetch\n");
#endif
}
/*-------------------------------------------------------------------------*/
global_operators_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_operators_store_delete\n");
  printf("Leaving global_operators_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
global_operators_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_operators_store_insert\n");
#endif

  Global_Operators = v;  /* Global_Operators is global variable */

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("leaving global_operators_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
global_types_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_types_store_init\n");
  printf("Leaving global_types_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE global_types_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_types_store_fetch\n");
  printf("Leaving global_types_store_fetch\n");
#endif
}
/*-------------------------------------------------------------------------*/
global_types_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_types_store_delete\n");
  printf("Leaving global_types_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
global_types_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_types_store_insert\n");
#endif

  Global_Types = v;  /* Global_Types is global variable */

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("leaving global_types_store_insert\n");
#endif
}

/*-------------------------------------------------------------------------*/
global_streams_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_streams_store_init\n");
  printf("Leaving global_streams_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE global_streams_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_streams_store_fetch\n");
  printf("Leaving global_streams_store_fetch\n");
#endif
}
/*-------------------------------------------------------------------------*/
global_streams_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_streams_store_delete\n");
  printf("Leaving global_streams_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
global_streams_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_streams_store_insert\n");
#endif
```

```
  Global_Streams = v;  /* Global_Streams is global variable */

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("leaving global_streams_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
global_states_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_states_store_init\n");
  printf("Leaving global_states_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE global_states_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_states_store_fetch\n");
  printf("Leaving global_states_store_fetch\n");
#endif
}
/*-------------------------------------------------------------------------*/
global_states_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_states_store_delete\n");
  printf("Leaving global_states_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
global_states_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_states_store_insert\n");
#endif

  Global_States = v;  /* Global_States is global variable */

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("leaving global_states_store_insert\n");
#endif
}

#endif /* CANCEL_TEMP_DELETE */
/*-------------------------------------------------------------------------*/
global_type_decl_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_type_decl_store_init\n");
  printf("Leaving global_type_decl_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE global_type_decl_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_type_decl_store_fetch\n");
  printf("Leaving global_type_decl_store_fetch\n");
#endif
}
/*-------------------------------------------------------------------------*/
global_type_decl_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_type_decl_store_delete\n");
  printf("Leaving global_type_decl_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
global_type_decl_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_type_decl_store_insert\n");
#endif

  Global_Type_Decl = v;  /* Global_Type_Decl is global variable */

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("leaving global_type_decl_store_insert\n");
#endif
}

/*-------------------------------------------------------------------------*/
#ifdef CANCEL_TEMP_DELETE

global_edges_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_edges_store_init\n");
  printf("Leaving global_edges_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE global_edges_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_edges_store_fetch\n");
  printf("Leaving global_edges_store_fetch\n");
#endif
```

```
}
/*-------------------------------------------------------------------------*/
global_edges_store_delete(p, a)
        PROD_INSTANCE p;
        ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_edges_store_delete\n");
  printf("Leaving global_edges_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
global_edges_store_insert(p, a, v)
        PROD_INSTANCE p;
        ATTR_NO a;
        PROD_INSTANCE v;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_edges_store_insert\n");
#endif

  Global_Edges = v;  /* Global_Edges is global variable */

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("leaving global_edges_store_insert\n");
#endif
}

#endif /* CANCEL_TEMP_DELETE */
/*-------------------------------------------------------------------------*/
global_undef_ops_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_undef_ops_store_init\n");
  printf("Leaving global_undef_ops_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE global_undef_ops_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_undef_ops_store_fetch\n");
  printf("Leaving global_undef_ops_store_fetch\n");
#endif
}
/*-------------------------------------------------------------------------*/
global_undef_ops_store_delete(p, a)
        PROD_INSTANCE p;
        ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_undef_ops_store_delete\n");
  printf("Leaving global_undef_ops_store_delete\n");
#endif

}
```

```
/*-------------------------------------------------------------------------*/
global_undef_ops_store_insert(p, a, v)
        PROD_INSTANCE p;
        ATTR_NO a;
        PROD_INSTANCE v;
{

PROD_INSTANCE
  temp_new_p,
  IsNull(),
  FirstElement(),
  IdSetTail(),
  Get_Id();

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_undef_ops_store_insert\n");
#endif

  Global_Undef_Ops = v;  /* Global_Undef_Ops is global variable */

#ifdef SDE_DEBUG_1

  if (v == NULLVALUE) return;

  temp_new_p = v;
  while(!BoolValue(IsNull(temp_new_p)))
  {
  printf("global_undef_ops_store_insert: id = %s\n",
       str0_to_str_ro(StrValue(Get_Id(FirstElement(temp_new_p)))));
  temp_new_p = IdSetTail(temp_new_p);
  }
  /* debugging */
  printf("leaving global_undef_ops_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
global_undef_type_ops_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_undef_type_ops_store_init\n");
  printf("Leaving global_undef_type_ops_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE global_undef_type_ops_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_undef_type_ops_store_fetch\n");
  printf("Leaving global_undef_type_ops_store_fetch\n");
#endif
}
/*-------------------------------------------------------------------------*/
global_undef_type_ops_store_delete(p, a)
        PROD_INSTANCE p;
        ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_undef_type_ops_store_delete\n");
  printf("Leaving global_undef_type_ops_store_delete\n");
```

```
#endif

}
/*-------------------------------------------------------------------------*/
global_undef_type_ops_store_insert(p, a, v)
        PROD_INSTANCE p;
        ATTR_NO a;
        PROD_INSTANCE v;
{

PROD_INSTANCE
  temp_new_p,
  IsNull(),
  FirstElement(),
  IdSetTail(),
  Get_Id();

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering global_undef_type_ops_store_insert\n");
#endif

  Global_Undef_Type_Ops = v;  /* Global_Undef_Type_Ops is global variable */

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("leaving global_undef_type_ops_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
operator_id_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering operator_id_store_init\n");
  printf("Leaving operator_id_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE operator_id_store_fetch()
{
       /* not used here */
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering operator_id_store_fetch\n");
  printf("Leaving operator_id_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
operator_id_store_delete(p, a)
        PROD_INSTANCE p;
        ATTR_NO a;
{

/*
  char
       *name;

  HeadPtr
       h,
       Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();
*/

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering operator_id_store_delete\n");
#endif

#ifdef CANCEL_TEMP_DELETE
  /* delete operators which have been marked for deletion */
  h = prototype;
  while (h != NULL)
  {
       if (h->marked_for_delete == TRUE)
       {
#ifdef SDE_DEBUG_2
               printf("REMOVING OPERATOR <<%s>> FROM LIST\n", h->name);
#endif

               WipeOutOperators(h->operator_list);
               WipeOutStreams(h->stream_list);
               RemoveHeadNode(h);
       }
       h = h->next;
  }

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h == NULL)
  {
       name = str0_to_str_ro(StrValue(Get_Id(id_from_Op(p))));
       printf("OPERATOR_STORE_DELETE: OPERATOR -->%s NOT FOUND IN STRUCTURE\n", name);
       /* the following statment is commented out, may cause memory leak */
       /* free(name); */
  }
#endif /*CANCEL_TEMP_DELETE */

#ifdef SDE_DEBUG_1
  printf("Leaving operator_id_store_delete\n");
#endif
}
/*-------------------------------------------------------------------------*/
operator_id_store_insert(p, a, v)
        PROD_INSTANCE p;
        ATTR_NO a;
        PROD_INSTANCE v;
{

  int
       prod_no;

  char
       *new_name;

  HeadPtr
       h,
       Make_Operator_Header(),
       Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  PROD_INSTANCE
       t;

  void
  Move_To_Structure_Front(),
  Link_To_Structure();
```

```
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering operator_id_store_insert\n");
#endif

  /* the following command is commented out for ease of debugging 1/21/94 */
  /* Output_Structure(); */

/*
  mark_killed_operators();
*/

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Operator_id_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  prod_no = (int)p;
  new_name = str0_to_str_ro(StrValue(Get_Id(v)));

#ifdef SDE_DEBUG_2
  /* debugging */
  printf("INSERT Operator %s %d\n", new_name, prod_no);
#endif

  if (h == NULL)
  {
        /* need to make new header node */
        h = Make_Operator_Header(new_name, NULL, NULL,
                                 Get_Unique_Id(), prod_no, FALSE);
        Link_To_Structure(h);
  }
  else
  {
        /* operator already in the list, update name and prod_no */
  if (strcmp(h->name, "") != 0)
  {
    free(h->name);
    free(h->ada_op_name);
  }
        h->name = strdup(new_name);
        h->ada_op_name = strdup(new_name);
        h->prod_no = h->prod_no;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }

  }

#ifdef SDE_DEBUG_1
  printf("Leaving operator_id_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
states_ids_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering states_ids_store_init\n");
  printf("Leaving states_ids_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE states_ids_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering states_ids_store_fetch\n");
  printf("Leaving states_ids_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
states_ids_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering states_ids_store_delete\n");
  printf("Leaving states_ids_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
states_ids_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
        h,
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering states_ids_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("States_ids_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
```

```
        h->state_id_set = v;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }

  }
  else
  {
        printf("states_ids_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving states_ids_store_insert\n");
#endif

}
/*-------------------------------------------------------------------------*/
inh_input_ids_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_input_ids_store_init\n");
  printf("Leaving inh_input_ids_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE inh_input_ids_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_input_ids_store_fetch\n");
  printf("Leaving inh_input_ids_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
inh_input_ids_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_input_ids_store_delete\n");
  printf("Leaving inh_input_ids_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
inh_input_ids_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
        h,
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_input_ids_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Inh_input_ids_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
        h->inh_input_id_set = v;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }

  }
  else
  {
        printf("inh_input_ids_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving inh_input_ids_store_insert\n");
#endif
}

/*-------------------------------------------------------------------------*/
inh_output_ids_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_output_ids_store_init\n");
  printf("Leaving inh_output_ids_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE inh_output_ids_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_output_ids_store_fetch\n");
  printf("Leaving inh_output_ids_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
inh_output_ids_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
```

```
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_output_ids_store_delete\n");
  printf("Leaving inh_output_ids_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
inh_output_ids_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
        h,
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_output_ids_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Inh_output_ids_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
        h->inh_output_id_set = v;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }

  }
  else
  {
        printf("inh_output_ids_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving inh_output_ids_store_insert\n");
#endif
}

/*-------------------------------------------------------------------------*/
input_error_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering input_error_store_init\n");
  printf("Leaving input_error_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE input_error_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering input_error_store_fetch\n");
  printf("Leaving input_error_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
input_error_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering input_error_store_delete\n");
  printf("Leaving input_error_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
input_error_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
  h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering input_error_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Input_error_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p)))));
  printf("Input_error_store_insert: Input_error = %s\n",
(BoolValue(v))?"true":"false");
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
  h->input_error = BoolValue(v);

  /* enforce most-recently-used-first rule */
  if (prototype != h)
```

```
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }
  }
  else
  {
  printf("input_error_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving input_error_store_insert\n");
#endif
}
/*--------------------------------------------------------------------------*/
output_error_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering output_error_store_init\n");
  printf("Leaving output_error_store_init\n");
#endif
}
/*--------------------------------------------------------------------------*/
PROD_INSTANCE output_error_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering output_error_store_fetch\n");
  printf("Leaving output_error_store_fetch\n");
#endif

}
/*--------------------------------------------------------------------------*/
output_error_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering output_error_store_delete\n");
  printf("Leaving output_error_store_delete\n");
#endif

}
/*--------------------------------------------------------------------------*/
output_error_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
  h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();



#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering output_error_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Output_error_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p)))));
  printf("Output_error_store_insert: Output_error = %s\n",
(BoolValue(v))?"true":"false");
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
  h->output_error = BoolValue(v);

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }
  }
  else
  {
  printf("output_error_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving output_error_store_insert\n");
#endif
}
/*--------------------------------------------------------------------------*/
inh_input_decl_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_input_decl_store_init\n");
  printf("Leaving inh_input_decl_store_init\n");
#endif
}
/*--------------------------------------------------------------------------*/
PROD_INSTANCE inh_input_decl_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_input_decl_store_fetch\n");
  printf("Leaving inh_input_decl_store_fetch\n");
#endif

}
/*--------------------------------------------------------------------------*/
inh_input_decl_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_input_decl_store_delete\n");
  printf("Leaving inh_input_decl_store_delete\n");
#endif
```

```
}
/*-------------------------------------------------------------------------*/
inh_input_decl_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
        h,
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_input_decl_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Inh_input_decl_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
        h->inh_input_decl = v;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }

  }
  else
  {
        printf("inh_input_decl_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving inh_input_decl_store_insert\n");
#endif
}

/*-------------------------------------------------------------------------*/
inh_output_decl_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_output_decl_store_init\n");
  printf("Leaving inh_output_decl_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE inh_output_decl_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_output_decl_store_fetch\n");
  printf("Leaving inh_output_decl_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
inh_output_decl_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_output_decl_store_delete\n");
  printf("Leaving inh_output_decl_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
inh_output_decl_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
        h,
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_output_decl_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Inh_output_decl_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
        h->inh_output_decl = v;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }

  }
  else
```

```
  {
        printf("inh_output_decl_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving inh_output_decl_store_insert\n");
#endif
}

/*-------------------------------------------------------------------------*/
met_error_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering met_error_store_init\n");
  printf("Leaving met_error_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE met_error_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering met_error_store_fetch\n");
  printf("Leaving met_error_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
met_error_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering met_error_store_delete\n");
  printf("Leaving met_error_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
met_error_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
  h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();



#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering met_error_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
```

```
  /* debugging */
  printf("Met_error_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p)))));
  printf("Met_error_store_insert: Met_error = %s\n", (BoolValue(v))?"true":"false");
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
  h->met_error = BoolValue(v);

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }
  }
  else
  {
  printf("met_error_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving met_error_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
inh_met_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_met_store_init\n");
  printf("Leaving inh_met_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE inh_met_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_met_store_fetch\n");
  printf("Leaving inh_met_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
inh_met_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_met_store_delete\n");
  printf("Leaving inh_met_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
inh_met_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
```

```
        PROD_INSTANCE v;
{
  HeadPtr
  h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();



#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering inh_met_store_insert\n");
#endif

  if (v == NULLVALUE) return;

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
  h->inh_met = v;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }
  }
  else
  {
  printf("inh_met_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving inh_met_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
impl_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering impl_store_init\n");
  printf("Leaving impl_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE impl_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering impl_store_fetch\n");
  printf("Leaving impl_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
impl_store_delete(p, a)
        PROD_INSTANCE p;
        ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering impl_store_delete\n");
  printf("Leaving impl_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
impl_store_insert(p, a, v)
        PROD_INSTANCE p;
        ATTR_NO a;
        PROD_INSTANCE v;
{
  PROD_INSTANCE
  Valid_Op_Impl();

  HeadPtr
  h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering impl_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Impl_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
  if (IntValue(v) < 2)
    h->is_composite = false;
  else
    h->is_composite = true;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }
  }
  else
  {
  printf("impl_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving impl_store_insert\n");
#endif
}
```

```
/*-------------------------------------------------------------------------*/
vertex_ids_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering vertex_ids_store_init\n");
  printf("Leaving vertex_ids_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE vertex_ids_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering vertex_ids_store_fetch\n");
  printf("Leaving vertex_ids_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
vertex_ids_store_delete(p, a)
        PROD_INSTANCE p;
        ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering vertex_ids_store_delete\n");
  printf("Leaving vertex_ids_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
vertex_ids_store_insert(p, a, v)
        PROD_INSTANCE p;
        ATTR_NO a;
        PROD_INSTANCE v;
{
  HeadPtr
       h,
       Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering vertex_ids_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Vertex_ids_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
       h->vertex_id_set = v;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }
  }
  else
  {
       printf("vertex_ids_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving vertex_ids_store_insert\n");
#endif

}
/*-------------------------------------------------------------------------*/
edge_ids_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering edge_ids_store_init\n");
  printf("Leaving edge_ids_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE edge_ids_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering edge_ids_store_fetch\n");
  printf("Leaving edge_ids_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
edge_ids_store_delete(p, a)
        PROD_INSTANCE p;
        ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering edge_ids_store_delete\n");
  printf("Leaving edge_ids_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
edge_ids_store_insert(p, a, v)
        PROD_INSTANCE p;
        ATTR_NO a;
        PROD_INSTANCE v;
{
  HeadPtr
       h,
       Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();
```

```
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering edge_ids_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Edge_ids_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
        h->edge_id_set = v;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }
  }
  else
  {
        printf("edge_ids_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving edge_ids_store_insert\n");
#endif

}
/*-----------------------------------------------------------------------------*/

#ifdef CANCEL_TEMP_DELETE

streams_ids_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering streams_ids_store_init\n");
  printf("Leaving streams_ids_store_init\n");
#endif
}
/*-----------------------------------------------------------------------------*/
PROD_INSTANCE streams_ids_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering streams_ids_store_fetch\n");
  printf("Leaving streams_ids_store_fetch\n");
#endif

}
/*-----------------------------------------------------------------------------*/
streams_ids_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering streams_ids_store_delete\n");
  printf("Leaving streams_ids_store_delete\n");
#endif

}
/*-----------------------------------------------------------------------------*/
streams_ids_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
        h,
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering streams_ids_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Stream_ids_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
       h->stream_id_set = v;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }

  }
  else
  {
        printf("streams_ids_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving streams_ids_store_insert\n");
#endif
}

/*-----------------------------------------------------------------------------*/
constraints_ids_store_init()
{
#ifdef SDE_DEBUG_1
```

```
  /* debugging */
  printf("Entering constraints_ids_store_init\n");
  printf("Leaving constraints_ids_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE constraints_ids_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering constraints_ids_store_fetch\n");
  printf("Leaving constraints_ids_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
constraints_ids_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering constraints_ids_store_delete\n");
  printf("Leaving constraints_ids_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
constraints_ids_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
        h,
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering constraints_ids_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Constraints_ids_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
        h->constraints_id_set  = v;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }
  }
  else
  {
        printf("constraints_ids_store_insert: operator not found\n");
  }


#ifdef SDE_DEBUG_1
  printf("Leaving constraints_ids_store_insert\n");
#endif

}

#endif /*CANCEL_TEMP_DELETE*/
/*-------------------------------------------------------------------------*/
multi_op_error_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering multi_op_error_store_init\n");
  printf("Leaving multi_op_error_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE multi_op_error_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering multi_op_error_store_fetch\n");
  printf("Leaving multi_op_error_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
multi_op_error_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering multi_op_error_store_delete\n");
  printf("Leaving multi_op_error_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
multi_op_error_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
  h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
```

```
  /* debugging */
  printf("Entering multi_op_error_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Multi_op_error_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p)))));
  printf("Multi_op_error_store_insert: Multi_op_error = %s\n",
(BoolValue(v))?"true":"false");
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
  h->multi_op_error = BoolValue(v);

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }
  }
  else
  {
  printf("multi_op_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving multi_op_error_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
stream_error_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering stream_error_store_init\n");
  printf("Leaving stream_error_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE stream_error_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering stream_error_store_fetch\n");
  printf("Leaving stream_error_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
stream_error_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering stream_error_store_delete\n");
  printf("Leaving stream_error_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
stream_error_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
  h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering stream_error_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Stream_error_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p)))));
  printf("Stream_error_store_insert: Stream_error = %s\n",
(BoolValue(v))?"true":"false");
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
  h->stream_error = BoolValue(v);

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }
  }
  else
  {
  printf("stream_error_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving stream_error_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
constraint_error_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering constraint_error_store_init\n");
  printf("Leaving constraint_error_store_init\n");
#endif
```

```
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE constraint_error_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering constraint_error_store_fetch\n");
  printf("Leaving constraint_error_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
constraint_error_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering constraint_error_store_delete\n");
  printf("Leaving constraint_error_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
constraint_error_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
  h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering constraint_error_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Constraint_error_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_Op(p)))));
  printf("Constraint_error_store_insert: Constraint_error = %s\n",
(BoolValue(v))?"true":"false");
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
  h->constraint_error = BoolValue(v);

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }
  }
  else
  {
  printf("constraint_error_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving constraint_error_store_insert\n");
#endif
}

/*-------------------------------------------------------------------------*/
t_multi_op_error_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_multi_op_error_store_init\n");
  printf("Leaving t_multi_op_error_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE t_multi_op_error_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_multi_op_error_store_fetch\n");
  printf("Leaving t_multi_op_error_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_multi_op_error_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_multi_op_error_store_delete\n");
  printf("Leaving t_multi_op_error_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_multi_op_error_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
  h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();

  char
  *new_name;

  int
  prod_no;
```

```
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_multi_op_error_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("T_multi_op_error_store_insert: Op Name = %s\n",
     str0_to_str_ro(StrValue(Get_Id(id_from_TOpSpec(p)))));
  printf("T_multi_op_error_store_insert: Multi_op_error = %s\n",
(BoolValue(v))?"true":"false");
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }
  }
  else
  {
        /* need to make new header node */

  new_name = str0_to_str_ro(StrValue(Get_Id(id_from_TOpSpec(p))));
  prod_no = (int)p;

        h = Make_Operator_Header(new_name, NULL, NULL,
                                 Get_Unique_Id(), prod_no, FALSE);
        Link_To_Structure(h);
  }

  h->multi_op_error = BoolValue(v);

#ifdef SDE_DEBUG_1
  printf("Leaving t_multi_op_error_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
t_type_id_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_type_id_store_init\n");
  printf("Leaving t_type_id_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE t_type_id_store_fetch()
{
        /* not used here */
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_type_id_store_fetch\n");
  printf("Leaving t_type_id_store_fetch\n");
#endif
}
/*-------------------------------------------------------------------------*/
t_type_id_store_delete(p, a)
        PROD_INSTANCE p;
        ATTR_NO a;
{

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_type_id_store_delete\n");
  printf("Leaving t_type_id_store_delete\n");
#endif
}
/*-------------------------------------------------------------------------*/
t_type_id_store_insert(p, a, v)
        PROD_INSTANCE p;
        ATTR_NO a;
        PROD_INSTANCE v;
{

  int
        prod_no;

  char
        *new_type_name;

  HeadPtr
        h,
        Make_Operator_Header(),
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  PROD_INSTANCE
        t;

  void
  Move_To_Structure_Front(),
  Link_To_Structure();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_type_id_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("T_type_id_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_TOpImpl(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
  if (strcmp(h->type_id, "") != 0)
  {
    free(h->type_id);
  }
        h->type_id = strdup(str0_to_str_ro(StrValue(Get_Id(v))));

  /* enforce most-recently-used-first rule */
```

```
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }

  }
  else
  {
        printf("t_type_id_store_insert: operator not found\n");
  }


#ifdef SDE_DEBUG_1
  printf("Leaving t_type_id_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
t_operator_id_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_operator_id_store_init\n");
  printf("Leaving t_operator_id_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE t_operator_id_store_fetch()
{
        /* not used here */
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_operator_id_store_fetch\n");
  printf("Leaving t_operator_id_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_operator_id_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{

/*
  char
        *name;

  HeadPtr
        h,
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();
*/

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_operator_id_store_delete\n");
#endif

#ifdef CANCEL_TEMP_DELETE
  /* delete operators which have been marked for deletion */
  h = prototype;
  while (h != NULL)
  {
        if (h->marked_for_delete == TRUE)
        {
#ifdef SDE_DEBUG_2
                printf("REMOVING OPERATOR <<%s>> FROM LIST\n", h->name);
#endif

                WipeOutOperators(h->operator_list);
                WipeOutStreams(h->stream_list);
                RemoveHeadNode(h);
        }
        h = h->next;
  }

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h == NULL)
  {
        name = str0_to_str_ro(StrValue(Get_Id(id_from_Op(p))));
        printf("OPERATOR_STORE_DELETE: OPERATOR -->%s NOT FOUND IN STRUCTURE\n", name);
        /* the following statment is commented out, may cause memory leak */
        /* free(name); */
  }
#endif /*CANCEL_TEMP_DELETE */

#ifdef SDE_DEBUG_1
  printf("Leaving t_operator_id_store_delete\n");
#endif
}
/*-------------------------------------------------------------------------*/
t_operator_id_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{

  int
        prod_no;

  char
        *new_name;

  HeadPtr
        h,
        Make_Operator_Header(),
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  PROD_INSTANCE
        t;

  void
  Move_To_Structure_Front(),
  Link_To_Structure();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_operator_id_store_insert\n");
#endif

  /* the following command is commented out for ease of debugging 1/21/94 */
  /* Output_Structure(); */

/*
  mark_killed_operators();
```

```
*/

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Operator_id_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_TOpImpl(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  prod_no = (int)p;
  new_name = str0_to_str_ro(StrValue(Get_Id(v)));

#ifdef SDE_DEBUG_2
  /* debugging */
  printf("INSERT Operator %s %d\n", new_name, prod_no);
#endif

  if (h == NULL)
  {
        /* need to make new header node */
        h = Make_Operator_Header(new_name, NULL, NULL,
                                 Get_Unique_Id(), prod_no, FALSE);
        Link_To_Structure(h);
  }
  else
  {
        /* operator already in the list, update name and prod_no */
  free(h->name);
  free(h->ada_op_name);
        h->name = strdup(new_name);
        h->ada_op_name = strdup(new_name);
        h->prod_no = h->prod_no;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }

  }

#ifdef SDE_DEBUG_1
  printf("Leaving t_operator_id_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
t_states_ids_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_states_ids_store_init\n");
  printf("Leaving t_states_ids_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE t_states_ids_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_states_ids_store_fetch\n");
  printf("Leaving t_states_ids_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_states_ids_store_delete(p, a)
        PROD_INSTANCE p;
        ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_states_ids_store_delete\n");
  printf("Leaving t_states_ids_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_states_ids_store_insert(p, a, v)
        PROD_INSTANCE p;
        ATTR_NO a;
        PROD_INSTANCE v;
{
  HeadPtr
        h,
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_states_ids_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("T_states_ids_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_TOpImpl(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
        h->state_id_set = v;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }

  }
  else
  {
        printf("t_states_ids_store_insert: operator not found\n");
  }
```

```
#ifdef SDE_DEBUG_1
  printf("Leaving t_states_ids_store_insert\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_inh_input_ids_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_inh_input_ids_store_init\n");
  printf("Leaving t_inh_input_ids_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE t_inh_input_ids_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_inh_input_ids_store_fetch\n");
  printf("Leaving t_inh_input_ids_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_inh_input_ids_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_inh_input_ids_store_delete\n");
  printf("Leaving t_inh_input_ids_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_inh_input_ids_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
        h,
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_inh_input_ids_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("T_inh_input_ids_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_TOpImpl(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
        h->inh_input_id_set = v;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }

  }
  else
  {
        printf("t_inh_input_ids_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving t_inh_input_ids_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
t_inh_output_ids_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_inh_output_ids_store_init\n");
  printf("Leaving t_inh_output_ids_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE t_inh_output_ids_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_inh_output_ids_store_fetch\n");
  printf("Leaving t_inh_output_ids_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_inh_output_ids_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_inh_output_ids_store_delete\n");
  printf("Leaving t_inh_output_ids_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_inh_output_ids_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
        h,
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();
```

```
  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_inh_output_ids_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("T_inh_output_ids_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_TOpImpl(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
        h->inh_output_id_set = v;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }

  }
  else
  {
        printf("t_inh_output_ids_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving t_inh_output_ids_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
t_inh_met_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_inh_met_store_init\n");
  printf("Leaving t_inh_met_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE t_inh_met_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_inh_met_store_fetch\n");
  printf("Leaving t_inh_met_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_inh_met_store_delete(p, a)
        PROD_INSTANCE p;
        ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_inh_met_store_delete\n");
  printf("Leaving t_inh_met_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_inh_met_store_insert(p, a, v)
        PROD_INSTANCE p;
        ATTR_NO a;
        PROD_INSTANCE v;
{
  HeadPtr
        h,
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_inh_met_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("T_inh_met_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_TOpImpl(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
        h->inh_met = v;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }

  }
  else
  {
        printf("t_inh_met_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving t_inh_met_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
t_impl_store_init()
{
#ifdef SDE_DEBUG_1
```

```
  /* debugging */
  printf("Entering t_impl_store_init\n");
  printf("Leaving t_impl_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE t_impl_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_impl_store_fetch\n");
  printf("Leaving t_impl_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_impl_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_impl_store_delete\n");
  printf("Leaving t_impl_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_impl_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  PROD_INSTANCE
  Valid_Op_Impl();

  HeadPtr
  h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_impl_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("T_impl_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_TOpImpl(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
  if (IntValue(v) < 2)
    h->is_composite = false;
  else
    h->is_composite = true;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }
  }
  else
  {
  printf("t_impl_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving t_impl_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
t_vertex_ids_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_vertex_ids_store_init\n");
  printf("Leaving t_vertex_ids_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE t_vertex_ids_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_vertex_ids_store_fetch\n");
  printf("Leaving t_vertex_ids_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_vertex_ids_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_vertex_ids_store_delete\n");
  printf("Leaving t_vertex_ids_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_vertex_ids_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
        h,
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();
```

```
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_vertex_ids_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("T_vertex_ids_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_TOpImpl(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
        h->vertex_id_set = v;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }
  }
  else
  {
        printf("t_vertex_ids_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving t_vertex_ids_store_insert\n");
#endif

}
/*--------------------------------------------------------------------------*/
t_edge_ids_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_edge_ids_store_init\n");
  printf("Leaving t_edge_ids_store_init\n");
#endif
}
/*--------------------------------------------------------------------------*/
PROD_INSTANCE t_edge_ids_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_edge_ids_store_fetch\n");
  printf("Leaving t_edge_ids_store_fetch\n");
#endif

}
/*--------------------------------------------------------------------------*/
t_edge_ids_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_edge_ids_store_delete\n");
  printf("Leaving t_edge_ids_store_delete\n");
#endif

}
/*--------------------------------------------------------------------------*/
t_edge_ids_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
        h,
        Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_edge_ids_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("T_edge_ids_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_TOpImpl(p)))));
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
        h->edge_id_set = v;

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }
  }
  else
  {
        printf("t_edge_ids_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving t_edge_ids_store_insert\n");
#endif

}
/*--------------------------------------------------------------------------*/
t_stream_error_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_stream_error_store_init\n");
  printf("Leaving t_stream_error_store_init\n");
```

```
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE t_stream_error_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_stream_error_store_fetch\n");
  printf("Leaving t_stream_error_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_stream_error_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_stream_error_store_delete\n");
  printf("Leaving t_stream_error_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_stream_error_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
  h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_stream_error_store_insert\n");
#endif

  if (v == NULLVALUE) return;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("T_stream_error_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_TOpImpl(p)))));
  printf("T_stream_error_store_insert: Stream_error = %s\n",
(BoolValue(v))?"true":"false");
#endif

  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
  h->stream_error = BoolValue(v);

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }
  }
  else
  {
  printf("t_stream_error_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving t_stream_error_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
t_constraint_error_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_constraint_error_store_init\n");
  printf("Leaving t_constraint_error_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE t_constraint_error_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_constraint_error_store_fetch\n");
  printf("Leaving t_constraint_error_store_fetch\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_constraint_error_store_delete(p, a)
         PROD_INSTANCE p;
         ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_constraint_error_store_delete\n");
  printf("Leaving t_constraint_error_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_constraint_error_store_insert(p, a, v)
         PROD_INSTANCE p;
         ATTR_NO a;
         PROD_INSTANCE v;
{
  HeadPtr
  h,
  Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec();

  void
  Move_To_Structure_Front();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_constraint_error_store_insert\n");
#endif

  if (v == NULLVALUE) return;
```

```
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("T_constraint_error_store_insert: Op Name = %s\n",
    str0_to_str_ro(StrValue(Get_Id(id_from_TOpImpl(p)))));
  printf("T_constraint_error_store_insert: Constraint_error = %s\n",
(BoolValue(v))?"true":"false");
#endif

  /* p is a t_op_impl production */
  h = Find_Header_Node_from_Op_or_TOpImpl_or_TOpSpec(p);

  if (h != NULL)
  {
  h->constraint_error = BoolValue(v);

  /* enforce most-recently-used-first rule */
  if (prototype != h)
  {
    /* move header_node to front of list */
    Move_To_Structure_Front(h);
  }
  }
  else
  {
  printf("t_constraint_error_store_insert: operator not found\n");
  }

#ifdef SDE_DEBUG_1
  printf("Leaving t_constraint_error_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
t_undefined_op_impl_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_undefined_op_impl_store_init\n");
  printf("Leaving t_undefined_op_impl_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE t_undefined_op_impl_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_undefined_op_impl_store_fetch\n");
  printf("Leaving t_undefined_op_impl_store_fetch\n");
#endif
}
/*-------------------------------------------------------------------------*/
t_undefined_op_impl_store_delete(p, a)
        PROD_INSTANCE p;
        ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_undefined_op_impl_store_delete\n");
  printf("Leaving t_undefined_op_impl_store_delete\n");
#endif

}
/*-------------------------------------------------------------------------*/
t_undefined_op_impl_store_insert(p, a, v)
        PROD_INSTANCE p;
        ATTR_NO a;
        PROD_INSTANCE v;
{

  PROD_INSTANCE
  Get_Id();

  TYPE_LIST
  tl,
  Find_Type_Node_from_Data(),
  Make_Type_Node();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_undefined_op_impl_store_insert\n");
#endif

  if (v == NULLVALUE) return;


  tl = Find_Type_Node_from_Data(p);
  if (tl == NULL)
  tl = Make_Type_Node(str0_to_str_ro(StrValue(Get_Id(id_from_Data(p)))));

  tl->undefined_op_impl = v;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("leaving t_undefined_op_impl_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
t_obsolete_op_impl_store_init()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_obsolete_op_impl_store_init\n");
  printf("Leaving t_obsolete_op_impl_store_init\n");
#endif
}
/*-------------------------------------------------------------------------*/
PROD_INSTANCE t_obsolete_op_impl_store_fetch()
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_obsolete_op_impl_store_fetch\n");
  printf("Leaving t_obsolete_op_impl_store_fetch\n");
#endif
}
/*-------------------------------------------------------------------------*/
t_obsolete_op_impl_store_delete(p, a)
        PROD_INSTANCE p;
        ATTR_NO a;
{
#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_obsolete_op_impl_store_delete\n");
  printf("Leaving t_obsolete_op_impl_store_delete\n");
#endif
```

```
}
/*-------------------------------------------------------------------------*/
t_obsolete_op_impl_store_insert(p, a, v)
        PROD_INSTANCE p;
        ATTR_NO a;
        PROD_INSTANCE v;
{

  PROD_INSTANCE
  Get_Id();

  TYPE_LIST
  tl,
  Find_Type_Node_from_Data(),
  Make_Type_Node();


#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering t_obsolete_op_impl_store_insert\n");
#endif

  if (v == NULLVALUE) return;


  tl = Find_Type_Node_from_Data(p);
  if (tl == NULL)
  tl = Make_Type_Node(str0_to_str_ro(StrValue(Get_Id(id_from_Data(p)))));

  tl->obsolete_op_impl = v;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("leaving t_obsolete_op_impl_store_insert\n");
#endif
}
/*-------------------------------------------------------------------------*/
int RemoveHeadNode(pointed_at)
  HeadPtr pointed_at;
  {
        extern HeadPtr
          prototype;
        HeadPtr
          p,
          q;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering RemoveHeadNode\n");
#endif

        q = NULL;
        p = prototype;
        while (p != pointed_at)
          {
                q = p;
                p = p->next;
          }

        if (q == NULL)
          {
                prototype = p->next;
          }
        else
          {
                q->next = p->next;
          }
        free(p);
  }
/*-------------------------------------------------------------------------*/
WipeOutSpline(s)
        SPLINE_PTR s;

{
  SPLINE_PTR last;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering WipeOutSpline\n");
#endif

  while (s != NULL)
        {
          last = s;
          s = s->next;
          free(last);
        }
}
/*-------------------------------------------------------------------------*/
WipeOutStreams(p)
        ST_PTR p;
{
  ST_PTR last;
  STREAM q;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering WipeOutStream\n");
#endif

  while (p != NULL)
        {
          last = p;
          q = p->st;
          WipeOutSpline(q->arc);
          free(q);
          p = p->next;
          free(last);
        }
#ifdef SDE_DEBUG_1
  printf("WipeOutStreams\n");
#endif
}
/*-------------------------------------------------------------------------*/
WipeOutOperators(p)
        OPNodePTR p;
{
  OPNodePTR last;
  OPERATOR q;

#ifdef SDE_DEBUG_1
  /* debugging */
  printf("Entering WipeOutOPerators\n");
#endif

  while (p != NULL)
```

```
        {
          last = p;
          q = p->op;
          free(q);
          p = p->next;
          free(last);
        }

#ifdef SDE_DEBUG_1
  printf("WipeOutOperators\n");
#endif
}
/*-------------------------------------------------------------------------*/
%]
/*-------------------------------------------------------------------------*/


/*************************************************************************
****  This eighth (last) set of functions were made in support of this thesis
****  and was named functions2.ssl
*************************************************************************/
/*
Application:  CAPS/SDE (Syntax Directed Editor)
System:       UNIX
Programmer:   Scott Grosenheider
LastTouched:  951205
Purpose:      Functions created in support of identifying simple
              timing and trigger constraints within a CAPS prototype
              so that simple errors can be identified and reported to
              the user while s/he is still in the editor, thereby
              reducing the time spent going back and forth from
              the scheduler to the editor.
*/


BOOL exported Is_Constrained_Operator(component c) {
   with(c) {
      NoComponent:        false,
      Data(i, ts, ti):    false,
      Op(i, os, oi):
      with(os) {
         OperatorSpec(*, *, *, *, *, opt_met, *, *, *):
         with(opt_met) {
            OpTimingInfoNone:     false,
            OpTimingInfoPrompt:   false,
            OpTimingInfo(met, *): true
         }
      }
   }
};
/* Do the constraints belong to a Constrained Operator (has a MET)? */
BOOL exported Belong2Constrained_Operator(a_constraint   ac,
                                          op_id_met_set  id_met_set) {
   with(ac)  {
      AConstraintNull:   false,
      AConstraint(op_id, *, *, *, *, *, *, *, *, *):
      Is_OpId_In_IdMetSet(op_id, id_met_set)
   }
};
BOOL Is_OpId_In_IdMetSet(operator_id    op_id,
                         op_id_met_set  id_met_set) {
   with(id_met_set)  {
      OpIdMetSetNil:   false,
      OpIdMetPair(hd, tl):
      with(hd)  {
         OpIdMetNull:  Is_OpId_In_IdMetSet(op_id, tl),
         OpIdMet(oid, *):
            (op_id == oid)
             ? true
             : Is_OpId_In_IdMetSet(op_id, tl)
      }
   }
};

/*-------------------------------------------------------------------------*/
per Get_Period(operator_id  opid, id_constraint_set cs)  {
   with(cs)  {
      IdConstraintSetNil:  PerNull,
      IdConstraintPair(hd, tl):
      with(hd)  {
         IdConstraintNull:  Get_Period(opid, tl),
         IdConstraint(oid, tc, *):
         (opid == oid)
         ? with(tc)  {
              OpConstraintsNull: PerNull,
              Periodic(per, *): per,
              Sporadic(*, *):   PerNull
           }
         : Get_Period(opid, tl)
      }
   }
};

BOOL exported
Is_ProducerOp_Period_LE_ConsumerOp(edge_set          es,
                                   id_constraint_set cs)  {
   with(es)  {
      EdgeSetNil:           false,
      EdgePair(hd, tl):
      with(hd)  {
         EdgeNull:         Is_ProducerOp_Period_LE_ConsumerOp(tl, cs),
         Edge(*, *, p, c):
         with(p)  {
            ProducerNull:     Is_ProducerOp_Period_LE_ConsumerOp(tl, cs),
            Producer(p_opid):
            with(c)  {
               ConsumerNull:     Is_ProducerOp_Period_LE_ConsumerOp(tl, cs),
               Consumer(c_opid):
               (Get_Period(p_opid, cs) == PerNull ||
                Get_Period(c_opid, cs) == PerNull    )
               ? Is_ProducerOp_Period_LE_ConsumerOp(tl, cs)
               : (Get_Period(p_opid, cs) <= Get_Period(c_opid, cs))
                 ? true
                 : Is_ProducerOp_Period_LE_ConsumerOp(tl, cs)
            }
         }
      }
   }
};

/*-------------------*/

BOOL Is_Sporadic(operator_id  opid, id_constraint_set cs)  {
   with(cs)  {
      IdConstraintSetNil: false,
      IdConstraintPair(hd, tl):
```

```
      with(hd)  (
         IdConstraintNull:          Is_Sporadic(opid, tl),
         IdConstraint(oid, tc, *):
         (oid == opid)
         ? with(tc)  (
              OpConstraintsNull:     false,
              Periodic(*, *):        false,
              Sporadic(*, *):        true
           )
         : Is_Sporadic(opid, tl)
      )
   )
};

trigger Get_Trigger(operator_id opid, id_constraint_set cs)  (
   with(cs)  (
      IdConstraintSetNil: TriggerByNull,
      IdConstraintPair(hd, tl):
      with(hd)  (
         IdConstraintNull:  Get_Trigger(opid, tl),
         IdConstraint(oid, *, trig):
         (opid == oid)
         ? trig
         : Get_Trigger(opid, tl)
      )
   )
};


BOOL exported
Is_Sporadic_ConsumerOp_WO_Trigger(edge_set          es,
                                  id_constraint_set cs)  (
   with(es)  (
      EdgeSetNil:           false,
      EdgePair(hd, tl):
      with(hd)  (
         EdgeNull:         Is_Sporadic_ConsumerOp_WO_Trigger(tl, cs),
         Edge(*, *, *, c):
         with(c)  (
            ConsumerNull:     Is_Sporadic_ConsumerOp_WO_Trigger(tl, cs),
            Consumer(c_opid):
            (Is_Sporadic(c_opid, cs))
            ? (Get_Trigger(c_opid, cs) == TriggerByNull)
              ? true
              : Is_Sporadic_ConsumerOp_WO_Trigger(tl, cs)
            : Is_Sporadic_ConsumerOp_WO_Trigger(tl, cs)
         )
      )
   )
};

/*-------------------*/

BOOL Is_Constrained_Opid(operator_id opid, op_id_met_set     ms)  (
   with(ms)  (
      OpIdMetSetNil:  false,
      OpIdMetPair(hd, tl):
      with(hd)  (
         OpIdMetNull:  Is_Constrained_Opid(opid, tl),
         OpIdMet(oid, met):
         (opid == oid)
         ? with(met) (
           MetNull: false,
           MET(*):  true
           )
         : Is_Constrained_Opid(opid, tl)
      )
   )
};

BOOL exported
Is_Constr_ProducerOp_And_Unconstr_ConsumerOp_W_Trigger(edge_set          es,
                                                       op_id_met_set     ms,
                                                       id_constraint_set cs)  (
   with(es)  (
      EdgeSetNil:           false,
      EdgePair(hd, tl):
      with(hd)  (
         EdgeNull:
         Is_Constr_ProducerOp_And_Unconstr_ConsumerOp_W_Trigger(tl, ms, cs),
         Edge(*, *, p, c):
         with(p)  (
            ProducerNull:
            Is_Constr_ProducerOp_And_Unconstr_ConsumerOp_W_Trigger(tl, ms, cs),
            Producer(p_opid):
            with(c)  (
               ConsumerNull:
               Is_Constr_ProducerOp_And_Unconstr_ConsumerOp_W_Trigger(tl, ms, cs),
               Consumer(c_opid):
               (   ( Is_Constrained_Opid(p_opid, ms)          )
                 &&(!Is_Constrained_Opid(c_opid, ms)          )
                 &&( Get_Trigger(c_opid, cs) != TriggerByNull) )
               ? true
               : Is_Constr_ProducerOp_And_Unconstr_ConsumerOp_W_Trigger(tl, ms, cs)
            )
         )
      )
   )
};

/*-------------------*/

BOOL exported
Is_Unconstr_ProducerOp_And_Constr_ConsumerOp_W_ByAll(edge_set          es,
                                                     op_id_met_set     ms,
                                                     id_constraint_set cs)  (
   with(es)  (
      EdgeSetNil:           false,
      EdgePair(hd, tl):
      with(hd)  (
         EdgeNull:
         Is_Unconstr_ProducerOp_And_Constr_ConsumerOp_W_ByAll(tl, ms, cs),
         Edge(*, *, p, c):
         with(p)  (
            ProducerNull:
            Is_Unconstr_ProducerOp_And_Constr_ConsumerOp_W_ByAll(tl, ms, cs),
            Producer(p_opid):
            with(c)  (
               ConsumerNull:
               Is_Unconstr_ProducerOp_And_Constr_ConsumerOp_W_ByAll(tl, ms, cs),
               Consumer(c_opid):
               (   (!Is_Constrained_Opid(p_opid, ms)          )
                 &&( Is_Constrained_Opid(c_opid, ms)          )
                 &&( Get_Trigger(c_opid, cs) == TriggerByAll) )
               ? true
               : Is_Unconstr_ProducerOp_And_Constr_ConsumerOp_W_ByAll(tl, ms, cs)
```

```
            )
         )
      )
   )
};


/*--------------------*/


BOOL exported
Is_Unconstr_ProducerOp_And_Constr_ConsumerOp_W_BySome(edge_set          es,
                                                      op_id_met_set     ms,
                                                      id_constraint_set cs)  {
   with(es)  {
      EdgeSetNil:           false,
      EdgePair(hd, tl):
      with(hd)  {
         EdgeNull:
         Is_Unconstr_ProducerOp_And_Constr_ConsumerOp_W_BySome(tl, ms, cs),
         Edge(*, *, p, c):
         with(p)  {
            ProducerNull:
            Is_Unconstr_ProducerOp_And_Constr_ConsumerOp_W_BySome(tl, ms, cs),
            Producer(p_opid):
            with(c)  {
               ConsumerNull:
               Is_Unconstr_ProducerOp_And_Constr_ConsumerOp_W_BySome(tl, ms, cs),
               Consumer(c_opid):
               (   (!Is_Constrained_Opid(p_opid, ms)        )
                 &&( Is_Constrained_Opid(c_opid, ms)        )
                 &&( Get_Trigger(c_opid, cs) == TriggerBySome) )
               ? true
               : Is_Unconstr_ProducerOp_And_Constr_ConsumerOp_W_BySome(tl, ms, cs)
            )
         )
      )
   )

};


/*-----------------------------------------------------------------------------*/
BOOL Is_Per(optional_period o_per)  {
   with(o_per)  {
      OptPeriodNull:        false,
      OptPeriodPrompt:      false,
      OptPeriod(time,*):    true
   )
};
BOOL Is_FW(optional_finish_within o_fw)  {
   with(o_fw)  {
      OptFinishWithinNull:       false,
      OptFinishWithinPrompt:     false,
      OptFinishWithin(time,*):   true
   )
};
BOOL Is_Mrt(optional_mrt o_mrt)  {
   with(o_mrt)  {
      OptMrtNull:           false,
      OptMrtPrompt:         false,
      OptMrt(time,*):       true
   )
};
BOOL Is_Mcp(optional_mcp o_mcp)  {
   with(o_mcp)  {
      OptMcpNull:        false,
      OptMcpPrompt:      false,
      OptMcp(time,*):    true
   )
};

/* Returns true if the constraint passed in has a time
   value for any of its four timing constraints.        */
BOOL exported Partial_Constraint(a_constraint ac)  {
   with(ac)  {
      AConstraintNull:      false,
      AConstraint(*, *, o_per, o_fw, o_mcp, o_mrt, *, *, *):
         ( Is_Per(o_per)  || Is_FW(o_fw) ||
           Is_Mrt(o_mrt) || Is_Mcp(o_mcp)  )
   )
};

/* Returns true if constraints are Periodic XOR Sporadic. */
BOOL exported Valid_T_Constraint(a_constraint ac)  {
   with(ac)  {
      AConstraintNull:      false,
      AConstraint(*, *, o_per, o_fw, o_mcp, o_mrt, *, *, *):
        Partial_Constraint(ac)  &&
        !( ( Is_Per(o_per) || Is_FW(o_fw)    )
                           &&
           ( Is_Mrt(o_mrt) || Is_Mcp(o_mcp) ) )
   )
};


/* Met <= FW < PER must exist for an Operator to be schedulable. */
BOOL exported Unschedulable_Periodic_Op(a_constraint  ac,
                                        op_id_met_set met_set) {
   with(ac)  {
      AConstraintNull:      false,
      AConstraint(op_id, *, o_per, o_fw, *, *, *, *, *):
      ( Is_Per(o_per) || Is_FW(o_fw) )
      ? with(Get_Met(op_id, met_set))  {
           MetNull:         false,
           MET(met_num):
           with(Get_Fw(o_fw))  {
              FwNull:
              with(Get_Per(o_per))  {
                 PerNull:  false,
                 Per(per_num): (met_num >= per_num)
              ),
              FW(fw_num):
              with(Get_Per(o_per))  {
                 PerNull:        (met_num > fw_num),
                 Per(per_num):  (   met_num >= fw_num
                                 || met_num >= per_num
                                 || fw_num  >= per_num)
              )
           )
         )
        : false
   )
};

/* 2*Met <= MRT <= 2*MCP must exist for an Operator to be schedulable. */
```

```
BOOL exported Unschedulable_Sporadic_Op(a_constraint  ac,
                                        op_id_met_set met_set) {
   with(ac)  {
      AConstraintNull:     false,
      AConstraint(op_id, *, *, *, o_mcp, o_mrt, *, *, *):
      ( Is_Mrt(o_mrt) || Is_Mcp(o_mcp) )
      ? with(Get_Met(op_id, met_set))  {
           MetNull:       false,
           MET(met_num):
           with(Get_Mcp(o_mcp))  {
              McpNull:
              with(Get_Mrt(o_mrt))  {
                 MrtNull:   false,
                 MRT(mrt_num): (2.0 * met_num > mrt_num)
              ),
              MCP(mcp_num):
              with(Get_Mrt(o_mrt))  {
                 MrtNull:       (met_num > mcp_num),
                 MRT(mrt_num): (   2.0 * met_num > mrt_num
                                || met_num > mcp_num
                                || mrt_num       > 2.0 * mcp_num)
              )
           )
        )
      : false
   )
};

BOOL exported Period_Only(a_constraint ac) {
   with(ac)  {
      AConstraintNull:     false,
      AConstraint(*, *, o_per, o_fw, o_mcp, o_mrt, *, *, *):
         ( Is_Per(o_per) && !Is_FW(o_fw) )
   )
};

BOOL exported FinishWithin_Only(a_constraint ac) {
   with(ac)  {
      AConstraintNull:     false,
      AConstraint(*, *, o_per, o_fw, o_mcp, o_mrt, *, *, *):
         ( !Is_Per(o_per) && Is_FW(o_fw) )
   )
};

BOOL exported MaxResTime_Only(a_constraint ac) {
   with(ac)  {
      AConstraintNull:     false,
      AConstraint(*, *, o_per, o_fw, o_mcp, o_mrt, *, *, *):
         ( Is_Mrt(o_mrt) && !Is_Mcp(o_mcp) )
   )
};

BOOL exported MinCallPeriod_Only(a_constraint ac) {
   with(ac)  {
      AConstraintNull:     false,
      AConstraint(*, *, o_per, o_fw, o_mcp, o_mrt, *, *, *):
         ( !Is_Mrt(o_mrt) && Is_Mcp(o_mcp) )
   )
};

REAL Smallest_Per(id_constraint_set cs, REAL  ans)  {
   with(cs)  {
      IdConstraintSetNil:      ans,
      IdConstraintPair(hd, tl):
      with(hd)  {
         IdConstraintNull:  Smallest_Per(tl, ans),
         IdConstraint(*, tc, *):
         with(tc)  {
            OpConstraintsNull: Smallest_Per(tl, ans),
            Periodic(per, *):
            with(per)  {
               PerNull:  Smallest_Per(tl, ans),
               Per(per_num):
               (per_num < ans)
               ? Smallest_Per(tl, per_num)
               : Smallest_Per(tl, ans)
            ),
            Sporadic(mcp, mrt):    /* <<<<<<<<<<===== THIS IS NOT CORRECT. */
               Smallest_Per(tl, ans)

         )
      )
   )
};

REAL Largest_Met(op_id_met_set  ms, REAL  ans) {
   with(ms)  {
      OpIdMetSetNil:     ans,
      OpIdMetPair(hd, tl):
      with(hd)  {
         OpIdMetNull:   Largest_Met(tl, ans),
         OpIdMet(*, m):
         with(m)   {
            MetNull:       Largest_Met(tl, ans),
            MET(met_num):
            (met_num > ans)
            ? Largest_Met(tl, met_num)
            : Largest_Met(tl, ans)
         )
      )
   )
};

BOOL exported Uniprocessor_Schedulable(op_id_met_set     ms,
                                       id_constraint_set cs) {
   Smallest_Per(cs, 1000000000000.0) > Largest_Met(ms, 0.0)
};

/* Does the actual work of getting # processors required.
   Called by Min_Processors_Required which converts to int. */
REAL Min_Processors_Required1(op_id_met_set     ms,
                              id_constraint_set cs) {
   with(cs)  {
      IdConstraintSetNil:   0.0,
      IdConstraintPair(hd, tl):
      with(hd)  {
         IdConstraintNull:     0.0 + Min_Processors_Required1(ms, tl),
         IdConstraint(oid, tc, *):
         with(tc)  {
            OpConstraintsNull: 0.0 + Min_Processors_Required1(ms, tl),
            Periodic(per, *):
            with(per)  {
               PerNull:       0.0 + Min_Processors_Required1(ms, tl),
               Per(per_num):
               with(Get_Met(oid, ms))  {
                  MetNull:    0.0 + Min_Processors_Required1(ms, tl),
```

```
                MET(met_num):
                (met_num / per_num) + Min_Processors_Required1(ms, tl)
             )
          ),
          Sporadic(mcp, mrt):    /* <<<<<<<<<<===== THIS IS NOT CORRECT. */
          0.0 + Min_Processors_Required1(ms, tl)

       )
    )
  )
);

/* Returns the # processors required by Cordeiro Dissertation pg 47.
   The .99 is used to approximate a ceiling function.                */
INT exported Min_Processors_Required(op_id_met_set     ms,
                                     id_constraint_set cs) {
  REALtoINT(Min_Processors_Required1(ms, cs) + 0.99)
};



/*- - - - - - - - */

id exported Extract_Id_From_IdConstraint(id_constraint idcons) {
  with(idcons)  {
     IdConstraintNull:       IdNull,
     IdConstraint(op_id, *, *):
     with(op_id)  {
        OperatorIdNull:       IdNull,
        OperatorId(*, id, *):   id
     }
  }
};

operator_id exported Extract_OpId_From_IdConstraint(id_constraint idcons) {
  with(idcons)  {
     IdConstraintNull:         OperatorIdNull,
     IdConstraint(op_id, *, *):  op_id
  }
};


id_constraint_set exported Id_Constraint_Set_Union(id_constraint_set s1,
                                    id_constraint_set s2) {
  with (s1) {
     IdConstraintSetNil: s2,
     IdConstraintPair(hd1, tl1):
     with(s2) {
        IdConstraintSetNil: s1,
        IdConstraintPair(hd2, tl2):
        (LessThanOpId(Extract_OpId_From_IdConstraint(hd1),
                      Extract_OpId_From_IdConstraint(hd2))
        ?  hd1 :: Id_Constraint_Set_Union(tl1, s2)
        :  (EqualOpId(Extract_OpId_From_IdConstraint(hd1),
                      Extract_OpId_From_IdConstraint(hd2))
           ?  hd1 :: Id_Constraint_Set_Union(tl1, tl2)
           :  hd2 :: Id_Constraint_Set_Union(s1, tl2)
           )
        )
     )
  )
};
```

```
per Get_Per(optional_period o_per)  {
   with(o_per)  {
      OptPeriodNull:      PerNull,
      OptPeriodPrompt:    PerNull,
      OptPeriod(t, *):
      (Convert_Time2Real(t) >= 0.0)
      ? Per(Convert_Time2Real(t))
      : PerNull
   }
};
fw Get_Fw(optional_finish_within o_fw)  {
   with(o_fw)  {
      OptFinishWithinNull:     FwNull,
      OptFinishWithinPrompt:   FwNull,
      OptFinishWithin(t, *):
      (Convert_Time2Real(t) >= 0.0)
      ? FW(Convert_Time2Real(t))
      : FwNull
   }
};
mcp Get_Mcp(optional_mcp o_mcp)  {
   with(o_mcp)  {
      OptMcpNull:      McpNull,
      OptMcpPrompt:    McpNull,
      OptMcp(t, *):
      (Convert_Time2Real(t) >= 0.0)
      ? MCP(Convert_Time2Real(t))
      : McpNull
   }
};
mrt Get_Mrt(optional_mrt o_mrt)  {
   with(o_mrt)  {
      OptMrtNull:      MrtNull,
      OptMrtPrompt:    MrtNull,
      OptMrt(t, *):
      (Convert_Time2Real(t) >= 0.0)
      ? MRT(Convert_Time2Real(t))
      : MrtNull
   }
};

met Get_Met(operator_id op_id, op_id_met_set met_set)  {
   with(met_set)  {
      OpIdMetSetNil:        MetNull,
      OpIdMetPair(hd, tl):
      with(hd)  {
         OpIdMetNull:        Get_Met(op_id, tl),
         OpIdMet(oid, the_met):
         (oid == op_id)
         ? the_met
         : Get_Met(op_id, tl)
      }
   }
};

/*  The default (assumed) unit associated w/ this int will be microseconds. */
REAL Convert_Time2Real(time t)  {
   with(t)  {
      TimeNull:          -1.0,
      Time(tv, tu):
      with(tv)  {
         IntegerNull:            -1.0,
```

```
        IntegerVal(str_digits):
        with(tu)  (
           UnitNil:          -1.0,
           UnitMICROSECONDS: INTtoREAL(STRtoINT(str_digits)),
           UnitMS:           (INTtoREAL(STRtoINT(str_digits)) *       1000.0),
           UnitSEC:          (INTtoREAL(STRtoINT(str_digits)) *    1000000.0),
           UnitMIN:          (INTtoREAL(STRtoINT(str_digits)) *   60000000.0),
           UnitHOURS:        (INTtoREAL(STRtoINT(str_digits)) * 3600000000.0)
        )
     )
  )
);


fw  Get_FW_from_Per(optional_period o_per)  (
  with(o_per)  (
     OptPeriodNull:      FwNull,
     OptPeriodPrompt:    FwNull,
     OptPeriod(t, *):
     (Convert_Time2Real(t) >= 0.0)
     ? FW(Convert_Time2Real(t))
     : FwNull
  )
);

per  Get_Per_from_Fw(optional_finish_within o_fw)  (
  with(o_fw)  (
     OptFinishWithinNull:     PerNull,
     OptFinishWithinPrompt:   PerNull,
     OptFinishWithin(t, *):
     (Convert_Time2Real(t) >= 0.0)
     ? Per(Convert_Time2Real(t))
     : PerNull
  )
);

mcp  Get_Mcp_from_Mrt(optional_mrt o_mrt, operator_id op_id, op_id_met_set  met_set)  (
  with(o_mrt)  (
     OptMrtNull:    McpNull,
     OptMrtPrompt:  McpNull,
     OptMrt(mrt_time, *):
     with(met_set)  (
        OpIdMetSetNil:   McpNull,
        OpIdMetPair(hd, tl):
        with(hd)  (
           OpIdMetNull:        Get_Mcp_from_Mrt(o_mrt, op_id, tl),
           OpIdMet(oid, met):
           (oid == op_id)
           ? with(met)  (
                MetNull:   McpNull,
                MET(met_num):
                (Convert_Time2Real(mrt_time) >= 0.0)
                ? MCP(Convert_Time2Real(mrt_time) - met_num)
                : McpNull
             )
           : Get_Mcp_from_Mrt(o_mrt, op_id, tl)
        )
     )
  )
);

mrt  Get_Mrt_from_Mcp(optional_mcp o_mcp, operator_id op_id, op_id_met_set met_set)  (
  with(o_mcp)  (
     OptMcpNull:          MrtNull,
     OptMcpPrompt:        MrtNull,
     OptMcp(mcp_time, *):
     with(met_set)  (
        OpIdMetSetNil:  MrtNull,
        OpIdMetPair(hd, tl):
        with(hd)  (
           OpIdMetNull:         Get_Mrt_from_Mcp(o_mcp, op_id, met_set),
           OpIdMet(oid, met):
           (oid == op_id)
           ? with(met)  (
                MetNull:  MrtNull,
                MET(met_num):
                (Convert_Time2Real(mcp_time) >= 0.0)
                ? MRT(met_num + Convert_Time2Real(mcp_time))
                : MrtNull
             )
           : Get_Mrt_from_Mcp(o_mcp, op_id, tl)
        )
     )
  )
);

trigger Insert_Trigger(optional_trigger ot)  (
  with(ot)  (
     OptionalTriggerNull:    TriggerByNull,
     OptionalTriggerPrompt:  TriggerByNull,
     OptionalTriggerAllOrSome(type_trigger, *, *, *):
     with(type_trigger)  (
        TriggerNull:   TriggerByNull,
        TriggerAll:    TriggerByAll,
        TriggerSome:   TriggerBySome
     ),
     OptionalIfExp(*, *):    TriggerByNull
  )
);

BOOL exported All_OR_Some_Trigger(a_constraint c)  (
  with(c)  (
     AConstraintNull: false,
     AConstraint(*, ot, *, *, *, *, *, *, *):
     with(ot)  (
        OptionalTriggerNull:   false,
        OptionalTriggerPrompt: false,
        OptionalTriggerAllOrSome(tt, *, *, *):
        with(tt)  (
           TriggerNull:  false,
           TriggerAll:   true,
           TriggerSome:  true
        )
     )
  )
);

id_constraint_set exported Get_Id_Constraint_Set(a_constraint  ac,
                                                 op_id_met_set met_set) (
  with(ac) (
     AConstraintNull:       IdConstraintSetNil,
     AConstraint(op_id, o_trigger, o_per, o_fw, o_mcp, o_mrt, *, *, *):
        (Valid_T_Constraint(ac))
         ? (Is_Per(o_per))
           ? (Is_FW(o_fw))
             ? IdConstraintPair
```

```
                (IdConstraint(op_id,
                              Periodic(Get_Per(o_per), Get_Fw(o_fw)),
                              Insert_Trigger(o_trigger)),
                 IdConstraintSetNil)
            : IdConstraintPair
                (IdConstraint(op_id,
                              Periodic(Get_Per(o_per), Get_FW_from_Per(o_per)),
                              Insert_Trigger(o_trigger)),
                 IdConstraintSetNil)
          : (Is_FW(o_fw))
            ? IdConstraintPair
                (IdConstraint(op_id,
                              Periodic(Get_Per_from_Fw(o_fw), Get_Fw(o_fw)),
                              Insert_Trigger(o_trigger)),
                 IdConstraintSetNil)
            : (Is_Mrt(o_mrt))
              ? (Is_Mcp(o_mcp))
                ? IdConstraintPair
                    (IdConstraint(op_id,
                                  Sporadic(Get_Mcp(o_mcp), Get_Mrt(o_mrt)),
                                  Insert_Trigger(o_trigger)),
                     IdConstraintSetNil)
                : IdConstraintPair
                    (IdConstraint(op_id,
                                  Sporadic(Get_Mcp_from_Mrt(o_mrt, op_id, met_set),
Get_Mrt(o_mrt)),
                                  Insert_Trigger(o_trigger)),
                     IdConstraintSetNil)
              : IdConstraintPair
                  (IdConstraint(op_id,
                                Sporadic(Get_Mcp(o_mcp), Get_Mrt_from_Mcp(o_mcp,
op_id, met_set)),
                                Insert_Trigger(o_trigger)),
                   IdConstraintSetNil)
          : ((Insert_Trigger(o_trigger) == TriggerByAll) ||
             (Insert_Trigger(o_trigger) == TriggerBySome)   )
            ? IdConstraintPair(IdConstraint(op_id,
                                            OpConstraintsNull,
                                            Insert_Trigger(o_trigger)),
                               IdConstraintSetNil)
            : IdConstraintSetNil
    )
);
/****************** commented ********************
id_constraint_set exported Get_Id_Constraint_Set(a_constraint  ac,
                                                 op_id_met_set met_set) {

   with(ac) (
      AConstraintNull:       IdConstraintSetNil,
      AConstraint(op_id, *, o_per, o_fw, o_mcp, o_mrt, *, *, *):
      IdConstraintSetNil
   )
);

****************** end test ********************/


/*- - - - - - - -  */

id exported Extract_Id_From_OpIdMet(op_id_met idmet) {
   with(idmet)  (
      OpIdMetNull:       IdNull,
      OpIdMet(op_id, constraints):
      with(op_id)  (
         OperatorIdNull:      IdNull,
         OperatorId(*, id, *):  id
      )
   )
);

operator_id exported Extract_OpId_From_OpIdMet(op_id_met idmet) {
   with(idmet)  (
      OpIdMetNull:                  OperatorIdNull,
      OpIdMet(op_id, constraints): op_id
   )
);


op_id_met_set exported Op_Id_Met_Set_Union(op_id_met_set s1, op_id_met_set s2) {
   with (s1) (
      OpIdMetSetNil: s2,
      OpIdMetPair(hd1, tl1):
      with(s2) (
         OpIdMetSetNil: s1,
         OpIdMetPair(hd2, tl2):
            (LessThanOpId(Extract_OpId_From_OpIdMet(hd1),
                          Extract_OpId_From_OpIdMet(hd2)))
            ?  hd1 :: Op_Id_Met_Set_Union(tl1, s2)
            :  (EqualOpId(Extract_OpId_From_OpIdMet(hd1),
                          Extract_OpId_From_OpIdMet(hd2)))
               ?  hd1 :: Op_Id_Met_Set_Union(tl1, tl2)
               :  hd2 :: Op_Id_Met_Set_Union(s1, tl2)
      )
   )
);

op_id_met_set exported Get_Id_Met_Set(a_vertex vertex) {
   with(vertex)  (
      AVertexNull:      OpIdMetSetNil,
      AVertex(op_id, opt_time):
      with(opt_time) (
         OptionalTimeNull:     OpIdMetSetNil,
         OptionalTimePrompt:   OpIdMetSetNil,
         OptionalTime(t):
         (Convert_Time2Real(t) >= 0.0)
         ? OpIdMetPair(OpIdMet(op_id, MET(Convert_Time2Real(t))),
                       OpIdMetSetNil)
         : OpIdMetPair(OpIdMet(op_id, MetNull), OpIdMetSetNil)
      )
   )
);

/*- - - - - - - -  */

/*
constrained_op_set exported Build_Constrained_op_Set(op_id_met_set      OpMetSet,
                                                     id_constraint_set IdConsSet)  (


);
*/

/*- - - - - - - -  */
edge_set exported Edge_Set_Union(edge_set s1, edge_set s2)  (
   with(s1) (
      EdgeSetNil: s2,
      EdgePair(hd1, tl1):
```

```
      with(s2)  (
         EdgeSetNil: s1,
         EdgePair(hd2, tl2):
         (LessThanOpId(Extract_OpId_From_Edge(hd1),
                       Extract_OpId_From_Edge(hd2))
         ? hd1 :: Edge_Set_Union(tl1, s2)
         : (EqualOpId(Extract_OpId_From_Edge(hd1),
                      Extract_OpId_From_Edge(hd2))
           ? hd1 :: Edge_Set_Union(tl1, tl2)
           : hd2 :: Edge_Set_Union(s1, tl2)
           )
         )
      )
   )
);

operator_id Extract_OpId_From_Edge(edge e)  (
   with(e)  (
      EdgeNull:   OperatorIdNull,
      Edge(edge_id, *, *, *): OperatorId(OptionalTypeIdNull,
                                         edge_id,
                                         OperatorIdPairsNull)
   )
);

edge_set exported Get_Edge_Set(an_edge e)  (
   with(e)  (
      AnEdgeNull:   EdgeSetNil,
      AnEdge(id, lt, fvi, tvi):
      with(fvi)  (
         FVertexIdNull:               EdgeSetNil,
         FVertexId(foti, fid, foip):
         with(tvi)  (
            TVertexIdNull:               EdgeSetNil,
            TVertexId(toti, tid, toip):
            with(lt)  (
               LatencyTimeNull:
                  EdgePair(Edge(id,
                                LatencyNull,
                                Producer(OperatorId(foti, fid, foip)),
                                Consumer(OperatorId(toti, tid, toip))),
                           EdgeSetNil),
               LatencyTimePrompt:
                  EdgePair(Edge(id,
                                LatencyNull,
                                Producer(OperatorId(foti, fid, foip)),
                                Consumer(OperatorId(toti, tid, toip))),
                           EdgeSetNil),
               LatencyTime(t):
                  EdgePair(Edge(id,
                                Latency(Convert_Time2Real(t)),
                                Producer(OperatorId(foti, fid, foip)),
                                Consumer(OperatorId(toti, tid, toip))),
                           EdgeSetNil),
            )
         )
      )
   )
);
```

APPENDIX E - Unparsing Rules

```
/* declare an output_view to output a clean PSDL program */
view SHOW_GRAPH_TEXT_VIEW, SDE_VIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW;

prototype
  : Prot[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::= error_header
       multiple_root_message multiple_root_ids
       multiple_op_spec_message multiple_op_spec
       multiple_type_spec_message multiple_type_spec
       also_op_type_message also_op_type_ids
       undefined_op_spec_message undefined_op_spec_set
       undefined_type_op_spec_message undefined_type_op_spec_set
       multiple_vertices_message multiple_vertices
       multiple_streams_message multiple_streams
          uniprocessor_unschedulability_msg
          min_processor_msg

       error_trailer @ ]
     [BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW @::= @]
  ;

psdl_components
  : PsdlNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ^::=]
  | PsdlPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
     IMPL_ONLY_VIEW  @::=^["%n"]@]
  ;

component
  : NoComponent[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="<COMPONENT LIST>"]
  | Op[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%nOPERATOR " @
         error_header
         root_message
         multiple_root_message
         multiple_op_spec_message
         also_defined_as_type_message
         undefined_input_message undefined_input_id
         obsolete_input_message obsolete_input_id
         undefined_output_message undefined_output_id
         obsolete_output_message obsolete_output_id
         input_type_error_message input_type_error_set
         output_type_error_message output_type_error_set
         obsolete_state_message obsolete_state_id
         met_error_message
         undefined_stream_message undefined_stream
         obsolete_stream_message obsolete_stream
         undefined_constraint_message undefined_constraint
         obsolete_constraint_message obsolete_constraint
         error_trailer @ @]
      [BASEVIEW @::="%nOPERATOR " @ @ @]
      [SPEC_ONLY_VIEW @::="%nOPERATOR " @ @ ..]
      [IMPL_ONLY_VIEW @::="%n" .. .. @]

  | Data[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%nTYPE "@
                                    error_header
                                    multiple_type_spec_message
                                    also_defined_as_op_message
         undefined_op_impl_message undefined_op_impl
         obsolete_op_impl_message obsolete_op_impl
                                    error_trailer @ @]
      [BASEVIEW @::="%nTYPE " @ @ @]
      [SPEC_ONLY_VIEW @::="%nTYPE " @ @ ..]
      [IMPL_ONLY_VIEW @::="%n" .. .. @]
  ;

id: IdNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ^::="<identifier>"]
      [BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW ^::="UNDEFINED_ID"]
  | Id[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW ^::=^]
  ;

integer: IntegerNull   [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::="<integer>"]
  | IntegerVal[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= ^]
  ;

type_spec
  : TypeSpec[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
     IMPL_ONLY_VIEW @::= "%t%nSPECIFICATION"
         @ /*o_generic_params*/
         @ /*o_type_decls*/
         @ /*o_operators*/
         @ /*o_keywords*/
         @ /*o_informal_descs*/
         @ /*o_formal_descs*/
         "%b%nEND"
     ]
  ;

o_generic_params
  : GenericNone[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | GenericPrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%t%n[optional generic
parameters]%b"]
  | Generic[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
     IMPL_ONLY_VIEW @::= "%t%nGENERIC"
          "%t%n" @ "%b%b"]/*type_declarations*/
  ;

type_declarations
  : TypeDeclNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | TypeDeclPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
     IMPL_ONLY_VIEW @::=^[",%n"]@]
  ;

a_decl: ADeclNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ^::="%t<type declaration>%b"]
  | ADecl[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
     IMPL_ONLY_VIEW ^::=
         @ " : " /*id_list*/
          @ ] /*undefined ADT decl_type_name*/
  ;

type_name
  : TypeNameNull  [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="<Type Name>"]
     [BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW @::= "UNDEFINED_TYPE_NAME"]
  | TypeName[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
     IMPL_ONLY_VIEW@::=   @/*type_identifier*/
          @]  /*o_bracket_type_declarations*/
  ;


decl_type_name
  : DTypeNameNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::= "<decl_type_name>"]
     [BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW @::= "UNDEFINED_TYPE_NAME"]
  | DTypeInteger[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
     IMPL_ONLY_VIEW @::= "INTEGER"]
```

```
  | DTypeReal[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW,  SPEC_ONLY_VIEW,
      IMPL_ONLY_VIEW  @::= "REAL"]
  | DTypeBoolean[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
      IMPL_ONLY_VIEW @::= "BOOLEAN"]
  | DTypeSimpleId[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
      IMPL_ONLY_VIEW @::= ^]
  | DTypeUserDefined[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
      IMPL_ONLY_VIEW @::=
        @ /* id */
        @]/* bracket_type_declarations */
  ;


o_bracket_type_declarations
  : OBTypeNone[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | OBTypePrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%n[type declarations]"]
  | OBTypeDeclaration[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW,
      SPEC_ONLY_VIEW, IMPL_ONLY_VIEW @::=
        "%n[%t%n"
        @ /*type_declarations*/
          "%b%n]"]
  ;

bracket_type_declarations
  : BTypeNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="[<type_declarations>]"]
  | BTypeDeclaration[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW,
      SPEC_ONLY_VIEW, IMPL_ONLY_VIEW @::=
        "%n[%t%n"
        ^ /*type_declarations*/
          "%b%n]"]
  ;

alone_id_list
  : AIdNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | AIdPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
      IMPL_ONLY_VIEW @::= ^ [","] @]
  ;

id_list
  : IdNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | IdPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::= ^ [multiply_defined ",%n"] @ ]
    [BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW @::= ^ [",%n"] @ ]
  ;

o_type_decls
  : TypeNone[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | TypePrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%t%n[optional type declarations]%b"]
  | Type[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW @::=
        "%t%n%n"
        @ "%b"]/*type_declarations*/
  ;

o_operators
  : OperatorNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | OperatorPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
      IMPL_ONLY_VIEW @::= ^ [" "] @]
  ;

t_oper_spec
  : TOpSpecNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ^::= "%t%n[optional operator]%b"]
  | TOpSpec[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ^::="%t%nOPERATOR "
            @ /*id*/
            error_header
            multiply_defined_message
            error_trailer
            @ "%b"]/*operator_spec*/
      [BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW  ^::="%t%nOPERATOR "
            @ /*id*/
            @ "%b"]/*operator_spec*/
  ;

operator_spec
  : OperatorSpec[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
      IMPL_ONLY_VIEW @::= "%t%nSPECIFICATION"
        @     /*o_generics_list*/
        @     /*o_inputs_list*/
        @     /*o_outputs_list*/
        @     /*o_states_list*/
        @     /*o_exceptions_list*/
        @     /*o_timing_info_list*/
        @     /*o_keywords*/
        @     /*o_informal_descs*/
        @     /*o_formal_descs*/
       "%b%nEND"]
  ;

o_generics_list
  : GenericsListNone[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | GenericsListPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
      IMPL_ONLY_VIEW @::= ^ @]
  ;

o_generics
  : OpGenericsNone[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%t%n[optional generics]%b"]
  | OpGenerics[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
      IMPL_ONLY_VIEW @::= "%t%nGENERIC"
          "%t%n" @/*type_declarations*/
              @ "%b%b"]/*reqmts_trace*/
  ;

o_inputs_list
  : InputsListNone[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | InputsListPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
      IMPL_ONLY_VIEW @::= ^ @]
  ;

o_inputs
  : OpInputsNone[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%t%n[optional inputs]%b"]
  | OpInputs[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
      IMPL_ONLY_VIEW @::= "%t%nINPUT"
          "%t%n"@/*type_declarations*/
          @ "%b%b"]/*reqmts_trace*/
  ;

o_outputs_list
  : OutputsListNone[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | OutputsListPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
      IMPL_ONLY_VIEW @::= ^ @]
  ;

o_outputs
  : OpOutputsNone[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%t%n[optional outputs]%b"]
  | OpOutputs[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
      IMPL_ONLY_VIEW @::= "%t%nOUTPUT"
          "%t%n"@/*type_declarations*/
          @ "%b%b"]/*requirements_trace*/
```

```
  ;

o_exceptions_list
  : ExcListNone[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | ExcListPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= ^ @]
  ;

o_exceptions
  : OpExceptionsNone[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%t%n[optional exceptions]%b"]
  | OpExceptions[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= "%t%nEXCEPTIONS"
            "%t%n"   @/*id_list*/
                  @ "%b%b"]/*reqmts_trace*/
  ;

o_timing_info
  : OpTimingInfoNone[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | OpTimingInfoPrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%t%n[optional timing
information]%b"]
  | OpTimingInfo[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=
            "%t%nMAXIMUM EXECUTION TIME "
            @    /*time*/
            "%t" @ "%b%b"] /*reqmts_trace*/
  ;


time
  : TimeNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::= "<time expression>"]
  | Time [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=   @/*integer*/
            @] /*time_unit*/
  ;

time_unit
  : UnitNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=" <units>"]
  | UnitMICROSECONDS[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=" MICROSEC"]
  | UnitMS[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=" MS"]
  | UnitSEC[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=" SEC"]
  | UnitMIN[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=" MIN"]
  | UnitHOURS[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=" HOURS"]
  ;

o_states_list
  : StatesListNone[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | StatesListPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= ^ @]
  ;

o_states
  : OpStatesNone  [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%t%n[optional states]%b"]
  | OpStates[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= "%t%nSTATES"
            "%t%n"@ /*type_declarations*/
            "%nINITIALLY"
            "%t%n"@ /*expression_list*/
                  @ "%b%b%b"]/*reqmts_trace*/
  ;

initial_args
  : InitialArgsNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | InitialArgs[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=  ^ ["%n"] @]
  ;

an_argument
  : AnArgNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::="<initial argument>"]
  | AnArgument[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::="(%t%n" @ "%b%n)"]
  ;


expression_list
  : InitialExpListNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | InitialExpListPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=^ [",%n"] @]
  ;

expression
  : ExpNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::="<exp>"]
  | Identifier[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= @]
  | Textual_Description[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= "\"" @ "\""]
  | TypeExpression[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= @ "." @   @ ]
  | ParenthesizedExp      [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= "(" @ ")"]

  /* BOOLEAN EXPRESSIONS  */

  | True [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::="TRUE"]
  | False[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::="FALSE"]
  | NotExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=" NOT " @]
  | EqualExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=@ " = " @]
  | LessExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=@ " < " @]
  | GreaterExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=@ " > " @]
  | GreatEqualExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=@ " >= " @]
  | LessEqualExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=@ " <= " @]
  | NotEquivExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=@ " /= " @]
  | AndExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=@ " AND " @]
  | OrExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=@ " OR " @]
  | XorExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=@ " XOR " @]

  /* ARITHMETIC EXPRESSIONS */
```

```
  | Integer[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=^]
  | Real [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=^ "." ^]
  | PlusExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " + " @]
  | MinusExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " - " @]
  | TimesExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " * " @]
  | DivExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " / " @]
  | NegativeExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=" - " @]
  | PositiveExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=" + " @]
  | AbsExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::="ABS(" @ ")"]
  | RemExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " rem " @]
  | ModExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " mod " @]
  | ExponentExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " ** " @]

  /* STRING EXPRESSIONS */

  | ConcatExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " & " @]
  ;

o_keywords
  : KeyWordsNone[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | KeyWordsPrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%t%n[optional keywords]%b"]
  | KeyWords[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::="%t%nKEYWORDS"
            "%t%n"@/*id_list*/
            "%b%b"]
  ;

o_informal_descs
  : InformalDescsNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | InformalDescsPrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%n[optional informal
description]"]
  | InformalDescs[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::="%nDESCRIPTION%t%n(" @ ")%b"]/*commentLines*/
  ;

o_formal_descs
  : FormalDescsNone[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | FormalDescsPrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%n[optional formal
description]"]
  | FormalDescs[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::="%nAXIOMS%t%n(" @ ")%b"]/*commentLines*/
  ;

reqmts_trace
  : ReqmtsTraceNone[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | ReqmtsTracePrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%n[requirements trace]"]
  | ReqmtsTrace[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::="%nREQUIRED BY"
            "%t%n"@ "%b"]/*id_list*/
  ;

text: TextNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="<text>"]
  | Text [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=^]
  ;

type_impl
  : TypeImplNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%n<type implementation>"]
  | AdaTypeImpl[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::="%nIMPLEMENTATION ADA " @ "%n%t" "%b%nEND"]
  | TypeImpl[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::="%nIMPLEMENTATION "
          @ /*type_name*/
          "%t"@ "%b%nEND"]/*operator_impl_list*/
  ;

operator_impl
  : OpImplNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%n<operator implementation>"]
  | AdaOpImpl   [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::="%nIMPLEMENTATION ADA " @ "%n%t" "%b%nEND"]

   | OperatorImpl
      [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%nIMPLEMENTATION"
            @     /*graph*/
        error_header
        producerop_period_le_consumerop_msg
        sporadic_consumerop_wo_trigger_msg
        constr_producerop_and_unconstr_consumerop_w_trigger_msg
        unconstr_producerop_and_constr_consumerop_w_byall_msg
        unconstr_producerop_and_constr_consumerop_w_bysome_msg
        error_trailer
            ^     /*declarations*/
            @     /*cc*/
          "%nEND"]

      [BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW @::= "%nIMPLEMENTATION"
            @     /*graph*/
            ^     /*declarations*/
            @     /*cc*/
          "%nEND"]
   ;

operator_impl_list
  : OpImplListNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | OpImplListPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::= ^ ["%n"] @]/*t_op_impl*/
  ;

t_op_impl
  : TOpImplNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ^::= "%t%n[operator implementation
list]%b"]
  | TOpImpl[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ^::= "%t%nOPERATOR "
              @   /*id*/
            error_header
            missing_op_spec_message
            multiply_defined_message
            obsolete_state_message obsolete_state_id_set
            undefined_stream_message undefined_stream
            obsolete_stream_message obsolete_stream
            undefined_constraint_message undefined_constraint
            obsolete_constraint_message obsolete_constraint
```

```
            error_trailer
            @ "%b"]/*operator_impl*/

      [BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW  ^::="%t%nOPERATOR "
            @ /*id*/
            @ "%b"]/*operator_impl*/
  ;

graph: GraphNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%t%n<graph>%b"]
  | Graph[SDE_VIEW @::="%t%nGRAPH%t%n-- see graph viewer for details --%b"
              ..          /*vertex_list*/
           "%n"
                  .. /*edge_list*/
            "%b"]
       [SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::="%t%nGRAPH"
             @ /*vertex_list*/
           "%n"
                  @ /*edge_list*/
            "%b"]
  ;

vertex_list
  :VertexListNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  |VertexListPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= ^ ["%n"] @]
  ;

a_vertex
  :AVertexNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ^::="%n[optional vertex list]"]
  |AVertex[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW ^::="%t%nVERTEX "
            @ /*operator_id*/
            @ /*optional_time*/
           "%b"]
  ;

operator_id
  : OperatorIdNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%n<operator id>"]
  | OperatorId[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=
           @ /*optional_type_id*/
           @ /*id*/
            @]/*operator_id_pairs*/
  ;

optional_type_id
  : OptionalTypeIdNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | OptionalTypeIdPrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="[optional type id] . "]
  | OptionalTypeId[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=
           @ "."/*id*/]
  ;


operator_id_pairs
  : OperatorIdPairsNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | OperatorIdPairsPrompt [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%n[optional vertex
pairs]%n"]
  | OperatorIdPairs[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::="(" @ "|"/*alone_id_list*/
              @ ")"]/*alone_id_list*/
  ;
```

```
optional_time
  : OptionalTimeNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | OptionalTimePrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="[optional time]"]
  | OptionalTime[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= " : " @ ]/*time*/
  ;

edge_list
  : EdgeListNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | EdgeListPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW  @::= ^ ["%n"] @]
  ;

an_edge
  : AnEdgeNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ^::= "%n[optional edge list]"]
  | AnEdge[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW ^::= "%t%nEDGE " @/*id*/
                    @/*latency_time*/
              "%t%n"
              @ " ->"  /*vertex_id*/
              "%n"
                  @ "%b%b"]/*vertex_id*/
  ;

latency_time
  : LatencyTimeNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | LatencyTimePrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="[optional latency time]"]
  | LatencyTime[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= " : " @ ]/*time*/
  ;

from_vertex_id
  : FVertexIdNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%n<vertex id>"]
       [BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW @::="%nUNDEFINED_ID"]
  | FVertexId[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=
           @    /*optional_type_name*/
           @    /*id*/
           @]   /*operator_id_pairs*/
  ;

to_vertex_id
  : TVertexIdNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%n<vertex id>"]
       [BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW @::="%nUNDEFINED_ID"]
  | TVertexId[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=
           @    /*optional_type_name*/
           @    /*id*/
           @]   /*operator_id_pairs*/
  ;

declarations
  : Declarations[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW ^::= @/*optional_streams*/
              @ ]/*optional_timers*/
  ;

optional_streams
  : StreamsNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | StreamsPrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%t%n[optional streams]%b"]
  | Streams[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::="%t%nDATA STREAM"
```

```
                "%t%n" @ /*type_declarations*/
                "%b%b"]
  ;

optional_timers
  : TimersNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | TimersPrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%t%n[optional timers]%b"]
  | Timers[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::="%t%nTIMER"
            "%t%n"@ "%b%b"]/*id_list*/
  ;

cc: CcNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::= "%t%n<control constraints>%b"]
  | Cc   [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::= "%t%nCONTROL CONSTRAINTS"
            @    /*constraints*/
            "%t"
            @ "%b%b"]/*o_informal_descs*/
  ;

constraints
  : ConstraintsNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | ConstraintsPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::= ^ ["%n"] @]
  ;

a_constraint
  : AConstraintNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::= "%t%n<constraint>%b"]
   | AConstraint     [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::= "%t%nOPERATOR "
                  @  /*id*/
               error_header
               unconstrained_op_with_constraints_msg
               invalid_time_constraint_msg
               unschedulable_periodic_msg
               unschedulable_sporadic_msg
               period_only_msg
               finish_within_only_msg
               mrt_only_msg
               mcp_only_msg
               error_trailer
                        @  /*optional_trigger*/
                     @  /*optional_period*/
                        @  /*optional_finish_within*/
                        @  /*optional_mcp*/
                        @  /*optional_mrt*/
                        @  /*output_guards*/
                        @  /*exception_ops*/
                        @"%b"]   /*timer_operations*/
                    [BASEVIEW, SPEC_ONLY_VIEW, IMPL_ONLY_VIEW @::= "%t%nOPERATOR "
            @ /*id*/
              @/*optional_trigger*/
            @ /*optional_period*/
              @/*optional_finish_within*/
              @/*optional_mcp*/
              @/*optional_mrt*/
              @/*output_guards*/
              @/*exception_ops*/
              @"%b"]/*timer_operations*/
  ;

optional_trigger
  : OptionalTriggerNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | OptionalTriggerPrompt [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%t%n[optional
trigger]%b"]
  | OptionalTriggerAllOrSome
        [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::="%t%nTRIGGERED BY "
            @ "%t%n"/*type_of_trigger*/
            @    /*id_list*/
            @    /*opt_if_predicate*/
            @ "%b%b"]/*reqmts_trace*/

  | OptionalIfExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW  @::="%t%nTRIGGERED IF%t%n"
            @
                    @ "%b%b"]/*reqmts_trace*/
  ;

type_of_trigger
  : TriggerNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="<trigger type>"]
  | TriggerAll[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::="ALL"]
  | TriggerSome[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::="SOME"]
  ;

optional_period
  : OptPeriodNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | OptPeriodPrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%t%n[optional period]%b"]
  | OptPeriod[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::="%t%nPERIOD "
            @ /*time*/
              "%t"    @/*reqmts_trace*/
              "%b%b"]
  ;

optional_finish_within
  : OptFinishWithinNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | OptFinishWithinPrompt [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%t%n[optional finish
within]%b"]
  | OptFinishWithin[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::="%t%nFINISH "
            @ /*time*/
              "%t" @/*reqmts_trace*/
             "%b%b"]
  ;

optional_mcp
  : OptMcpNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | OptMcpPrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%t%n[optional Min. Calling
Period]%b"]
  | OptMcp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::="%t%nMINIMUM CALLING PERIOD "
              @/*time*/
             "%t"   @ "%b%b"]/*reqmts_trace*/
  ;

optional_mrt
  : OptMrtNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | OptMrtPrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%t%n[optional Max. Resp. Time]%b"]
  | OptMrt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::="%t%nMAXIMUM RESPONSE TIME "
              @/*time*/
             "%t"   @ "%b%b"]/*reqmts_trace*/
  ;
```

APPENDIX E - Unparsing Rules

```
output_guards
  : OutputGuardsNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | OutputGuardsPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= ^ ["%n"] @]
  ;

a_guard
  : AGuardNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%t%n[output guard]%b"]
  | AGuard[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::="%t%nOUTPUT"
         "%t%n"@/*id_list*/
              "%nIF " @/*c_expression*/
                @ "%b%b"]/*reqmts_trace*/
  ;

exception_ops
  : ExceptionOpsNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ^::=]
  | ExceptionOpsPrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ^::="%t%n[optional
exceptions]%b"]
  | Exception[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW ^::= @]
  ;

exception_options
  : ExceptionOptionsNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | ExceptionOptionsPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= ^ ["%n"] @]
  ;

an_exception
  : AnExceptionNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ^::="%t%n<an exception>%b"]
  | AnException[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW ^::="%t%nEXCEPTION "
            @/*id*/
          "%t   " @/*optional_if_predicate*/
              @ "%b%b"]/*reqmts_trace*/
  ;
timer_operations
  : TimerOperationsNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | TimerOperationsPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= ^ ["%n"] @]
  ;

a_timer_operation
  : ATimerOperationNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ^::="%t%n[timer operation]%b"]
  | ATimerReset[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW ^::= "%t%nRESET " @ /*id*/
           "%t"     @/*optional_if_predicate*/
                    @ "%b%b"]/*reqmts_trace*/

  | ATimerStart[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= "%t%nSTART "
           @ /*id*/
           "%t"  @/*optional_if_predicate*/
                 @ "%b%b"]/*reqmts_trace*/

  | ATimerStop[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= "%t%nSTOP "
           @ /*id*/
           "%t"  @/*optional_if_predicate*/
                 @ "%b%b"]/*reqmts_trace*/
  ;
```

```
optional_if_predicate
  : OptIfPredicateNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | OptIfPredicatePrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="%n[optional IF
predicate]"]
  | OptIfPredicate[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= "%nIF "
           @]/*c_expression*/
  ;

/*   Conditionals for IF expressions    */

c_initial_args
  : CInitialArgsNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | CInitialArgs[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= ^ ["%n"] @ ]
  ;

c_an_argument
  : CAnArgNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::="<initial argument>"]
  | CAnArgument[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::="(%t%n" @ "%b%n)"]
  ;

c_expression_list
  : CInitialExpListNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::=]
  | CInitialExpListPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=^ [",%n"] @]
  ;

c_expression
  : CExpNull[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::="<exp>"]
  | CIdentifier[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= @]
  | CTextual_Description[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= "\"" @ "\""]
  | CTypeExpression[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= @ "." @ @ ]
  | CTimeExpression[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= @]
  | CParenthesizedExp     [SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::= "(" @ ")"]

  /* BOOLEAN EXPRESSIONS */

  | CTrue[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::="TRUE"]
  | CFalse[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::="FALSE"]
  | CNotExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=" NOT " @]
  | CEqualExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=@ " = " @]
  | CLessExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=@ " < " @]
  | CGreaterExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=@ " > " @]
  | CGreatEqualExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=@ " >= " @]
  | CLessEqualExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
       IMPL_ONLY_VIEW @::=@ " <= " @]
  | CNotEquivExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
```

```
        IMPL_ONLY_VIEW @::=@ " /= " @]
  | CAndExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " AND " @]
  | COrExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " OR " @]
  | CXorExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " XOR " @]

  /* INTEGER EXPRESSIONS */

  | CInteger[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=^]
  | CReal[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=^ "." ^]
  | CPlusExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " + " @]
  | CMinusExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " - " @]
  | CTimesExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " * " @]
  | CDivExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " / " @]
  | CNegativeExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=" - " @]
  | CPositiveExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=" + " @]
  | CAbsExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::="ABS(" @ ")"]
  | CRemExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " rem " @]
  | CModExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " mod " @]
  | CExponentExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " ** " @]

  /* STRING EXPRESSIONS */

  | CConcatExp[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=@ " & " @]
  ;


/*------unformatted text  ----------*/

commentLines
  : CommentLinesNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @:]
  | CommentLinesPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW @::=^["%n "]@]
  ;

commentLine
  : CommentLineNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ^:"<unformatted text>"]
  | CommentLine[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW, BASEVIEW, SPEC_ONLY_VIEW,
        IMPL_ONLY_VIEW ^:^]
  ;

/******************
optionalComment
  : OptionalCommentNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW ^:]
  | OptionalCommentPrompt[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="<unformatted text>"]
  | OptionalComment[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @::="("@")"]
  ;
*******************/
```

```
/*------IdSet  ----------*/
IdSet
  : IdSetNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @:]
  | IdSetPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @: "--      " @ [",%n"]@]
  ;


/*------OpIdSet  ----------*/
OpIdSet
  : OpIdSetNil[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @:]
  | OpIdSetPair[SDE_VIEW, SHOW_GRAPH_TEXT_VIEW @: "--      " @ [",%n"]@]
  ;
```

APPENDIX F - Transformation Rules

```
transform component
  on "Type"<component>:Data(
          <id>,
          <type_spec>,
          <type_impl>
          ),

  on "Operator" <component>:Op(
          <id>,
          <operator_spec>,
          <operator_impl>
          )

  ;

transform type_spec
  on "TypeSpec" <type_spec>:TypeSpec(
          <o_generic_params>,
          <o_type_decls>,
          <o_operators>,
          <o_keywords>,
          <o_informal_descs>,
          <o_formal_descs>
          )
  ;

transform o_generic_params
  on "Generic" <o_generic_params>:Generic(
          <type_declarations>
          )
  ;

transform o_type_decls
  on "Type"<o_type_decls>:Type(
          <type_declarations>
          )
  ;

transform t_oper_spec
  on "Operators"<t_oper_spec>:TOpSpec(
          <id>,
          <operator_spec>
          )
  ;


transform time_unit
  on "MICROSEC"<time_unit>: UnitMICROSECONDS(),
  on "MS"<time_unit>: UnitMS(),
  on "SECONDS"<time_unit>: UnitSEC(),
  on "MINUTES"<time_unit>: UnitMIN(),
  on "HOURS"      <time_unit>: UnitHOURS()
  ;

transform o_keywords
  on "Keywords"<o_keywords>:KeyWords(
          <alone_id_list>
          )
  ;

transform o_informal_descs
  on "Informal_Descriptions"<o_informal_descs>: InformalDescs(
          <commentLines>
          )
  ;

transform o_formal_descs
  on "Formal_Descriptions"
       <o_formal_descs>: FormalDescs(
            <commentLines>
            )
  ;

transform operator_spec
  on "Operator"<operator_spec>: OperatorSpec(
          <o_generics_list>,
          <o_inputs_list>,
          <o_outputs_list>,
          <o_states_list>,
          <o_exceptions_list>,
          <o_timing_info>,
          <o_keywords>,
          <o_informal_descs>,
          <o_formal_descs>
          )
  ;

transform o_generics
  on "OptionalGenerics"
       <o_generics>:OpGenerics(
            <type_declarations>,
            <reqmts_trace>
            )
  ;

transform o_inputs
  on "OptionalInputs"
       <o_inputs>: OpInputs(
            <type_declarations>,
            <reqmts_trace>
            )
  ;

transform o_outputs
  on "OptionalOutputs"
       <o_outputs>: OpOutputs(
            <type_declarations>,
            <reqmts_trace>
            )
  ;

transform o_states
  on "OptionalStates"
       <o_states>: OpStates(
            <type_declarations>,
            <expression_list>,
            <reqmts_trace>
            )
  ;

transform an_argument
  on "Exp_Argument"
       <an_argument>: AnArgument(
            <expression_list>
               )
  ;
```

APPENDIX F - Transformation Rules

```
transform o_exceptions
  on "Optional_Exceptions"
       <o_exceptions>: OpExceptions(
           <alone_id_list>,
           <reqmts_trace>
           )
  ;

transform o_timing_info
  on "Optional_Timing_Info"
       <o_timing_info>: OpTimingInfo(
           <time>,
           <reqmts_trace>
           )
  ;

transform time
  on "Time_Expression"
       <time>: Time(
         <integer>,
         <time_unit>
         )
  ;


transform reqmts_trace
  on "ReqmtsTrace"
       <reqmts_trace>: ReqmtsTrace(
           <alone_id_list>
           )
  ;


/*
--------TYPE DECLARATION TRANSFORMATIONS
--------
--------
--------
*/

transform a_decl
  on "DeclList"<a_decl>:ADecl(
         <id_list>,
         <decl_type_name>
         )
  ;

transform type_name
  on "TypeName"<type_name>: TypeName(
         <id>,
         <o_bracket_type_declarations>
         )
  ;

transform decl_type_name
  on "INTEGER"<decl_type_name>: DTypeInteger(),
  on "FLOAT"<decl_type_name>: DTypeSimpleId(Id("FLOAT")),
  on "BOOLEAN"<decl_type_name>: DTypeBoolean(),
/*
  on "EXCEPTION"<decl_type_name>: DTypeException(),
*/
  on "EXCEPTION"<decl_type_name>: DTypeSimpleId(Id("PSDL_EXCEPTION")),
  on "User_Defined"
       <decl_type_name>: DTypeUserDefined(
             <id>,
             <bracket_type_declarations>
             ),
  on "REAL"<decl_type_name>: DTypeReal()
  ;


transform o_bracket_type_declarations
  on "[Type Declarations]"
    <o_bracket_type_declarations>:OBTypeDeclaration(
         <type_declarations>
         )
  ;

transform bracket_type_declarations
  on "Type Declarations"
    <bracket_type_declarations>:BTypeDeclaration(
         <type_declarations>
         )
  ;

/*
-----------     EXPRESSION TEMPLATE TRANSFORMATIONS
*/

transform expression
  on "Ident"<expression>: Identifier(<id>),
  on "TypeExpresion"
       <expression>: TypeExpression(
             <type_name>,
             <id>,
             <initial_args>
                 ),

  on "Textual"<expression>: Textual_Description(<commentLines>),
/*
  on "Textual"<expression>: Textual_Description(<string_lit>),
*/

  on "(<expression>)"
       <expression>: ParenthesizedExp(<expression>),

  /* boolean_expression */

  on "And"<expression>: AndExp(
             <expression>,
             <expression>
             ),

  on "Or"<expression>: OrExp(
             <expression>,
             <expression>
             ),

  on "Xor"<expression>: XorExp(
             <expression>,
             <expression>
             ),

  on "True"<expression>: True(),
```

APPENDIX F - Transformation Rules

```
on "False"<expression>: False(),

on "Not"<expression>: NotExp(
          <expression>
          ),

on "="<expression>: EqualExp(
          <expression>,
          <expression>
          ),

on "<"<expression>: LessExp(
          <expression>,
          <expression>
          ),

on ">"<expression>: GreaterExp(
          <expression>,
          <expression>
          ),

on ">="<expression>: GreatEqualExp(
          <expression>,
          <expression>
          ),

on "<="<expression>: LessEqualExp(
          <expression>,
          <expression>
          ),

on "/="<expression>: NotEquivExp(
          <expression>,
          <expression>
          ),

/* arithmetic_expression */

on "Integer"<expression>: Integer(<integer>),

on "Real"<expression>: Real(<integer>, <integer>),

on "+"<expression>: PlusExp(
          <expression>,
          <expression>
          ),

on "-"<expression>: MinusExp(
          <expression>,
          <expression>
          ),

on "*"<expression>: TimesExp(
          <expression>,
          <expression>
          ),

on "/"<expression>: DivExp(
          <expression>,
          <expression>
          ),

  on "Negative"<expression>: NegativeExp(
            <expression>
            ),

  on "Mod"<expression>: ModExp(
            <expression>,
            <expression>
            ),

  on "Rem"<expression>: RemExp(
            <expression>,
            <expression>
            ),

  on "**"<expression>: ExponentExp(
            <expression>,
            <expression>
            ),

  /* string expressions */
  on "&"<expression>: ConcatExp(
            <expression>,
            <expression>
            )
  ;


/*TEMPLATE TRANSFORMATIONS FOR NEW PRODUCTIONS
*/

transform type_impl
  on "Ada_Implementation"
      <type_impl>:Get_Ada_Type_Impl(),
/*
            <id>
                ),
*/

  on "PSDL_Implementation"
      <type_impl>:TypeImpl(
            <type_name>,
                <operator_impl_list>
              )
  ;

transform t_op_impl
  on "Operator_Impl"
      <t_op_impl>:TOpImpl(
            <id>,
                 <operator_impl>
             )
  ;

transform operator_impl
  on "Ada_Implementation"<operator_impl>:Get_Ada_Op_Impl(),
/*
              <id>
             ),
*/
  on "PSDL_Implementation"
       <operator_impl>:OperatorImpl(
            <graph>,
            <declarations>,
```

```
            <cc>
            )
  ;

transform graph
/*
       on "Graph-Edit" <graph>:Edit_Graph(),
       on "Graph-Edit" Graph(vertex_list, edge_list):Edit_Graph(),
*/

  on "Graph-Decl"<graph> when (Is_Show_Graph_Text_View())
       :Graph(
         <vertex_list>,
         <edge_list>
           )
  ;

transform a_vertex
  on "Vertex"<a_vertex>:AVertex(
         <operator_id>,
         <optional_time>
         )
  ;

transform operator_id
  on "Operator_IDs"
     <operator_id>:OperatorId(
           <optional_type_id>,
           <id>,
           <operator_id_pairs>
           )
  ;

transform operator_id_pairs
  on "OperatorPairs"
     <operator_id_pairs>:OperatorIdPairs(
               <alone_id_list>,
               <alone_id_list>
               )
  ;

transform optional_time
  on "TimeEntry"<optional_time>:OptionalTime(
           <time>
           )
  ;


transform an_edge
  on "EdgeEntry"<an_edge>:AnEdge(
         <id>,
         <latency_time>,
         <from_vertex_id>,
         <to_vertex_id>
         )
  ;

transform latency_time
  on "LatencyTime" <latency_time>:LatencyTime(
           <time>
         )
  ;
```

```
transform from_vertex_id
  on "Vertex_ID"<from_vertex_id>:FVertexId(
         <optional_type_id>,
         <id>,
         <operator_id_pairs>
         )
  ;

transform to_vertex_id
  on "Vertex_ID"<to_vertex_id>:TVertexId(
         <optional_type_id>,
         <id>,
         <operator_id_pairs>
         )
  ;

transform declarations
  on "Declarations"
     <declarations>:Declarations(
         <optional_streams>,
         <optional_timers>
         )
  ;

transform optional_streams
  on "Streams"<optional_streams>:Streams(
         <type_declarations>
         )
  ;

transform optional_timers
  on "Timers"<optional_timers>:Timers(
           <alone_id_list>
           )
  ;

transform cc
  on "ControlConstraints"
     <cc>:Cc(
       <constraints>,
       <o_informal_descs>
       )
  ;
transform a_constraint
  on "A_Constraint"
     <a_constraint>:AConstraint(
         <operator_id>,
         <optional_trigger>,
         <optional_period>,
         <optional_finish_within>,
         <optional_mcp>,
         <optional_mrt>,
         <output_guards>,
         <exception_ops>,
         <timer_operations>
         )
  ;

transform optional_trigger
  on "ALL/SOME-Trigger"
     <optional_trigger>:OptionalTriggerAllOrSome(
         <type_of_trigger>,
         <alone_id_list>,
```

APPENDIX F - Transformation Rules

```
            <optional_if_predicate>,
            <reqmts_trace>
            ),

  on "IF-Expression"
      <optional_trigger>:OptionalIfExp(
            <expression>,
            <reqmts_trace>
            )
  ;
transform type_of_trigger
  on "ALL-Trigger"<type_of_trigger>:TriggerAll(),
  on "SOME-Trigger"<type_of_trigger>:TriggerSome()
  ;

transform optional_period
  on "Optional_Period"<optional_period>:OptPeriod(
              <time>,
              <reqmts_trace>
              )
  ;

transform optional_finish_within
  on "FinishWithin"
      <optional_finish_within>:OptFinishWithin(
              <time>,
              <reqmts_trace>
              )
  ;

transform optional_mrt
  on "MaxRespTime"
      <optional_mrt>:OptMrt(
              <time>,
              <reqmts_trace>
              )
  ;

transform optional_mcp
  on "MinCallPeriod"
      <optional_mcp>:OptMcp(
              <time>,
              <reqmts_trace>
              )
  ;

transform a_guard
  on "OutputGuard"
      <a_guard>:AGuard(
            <alone_id_list>,
            <c_expression>,
            <reqmts_trace>
            )
  ;

transform exception_ops
  on "Exception"<exception_ops>:Exception(
              <exception_options>
              )
  ;

transform an_exception
  on "AnException"
      <an_exception>:AnException(
            <id>,
            <optional_if_predicate>,
            <reqmts_trace>
            )
  ;

transform a_timer_operation
  on "Reset"<a_timer_operation>:ATimerReset(
            <id>,
            <optional_if_predicate>,
            <reqmts_trace>
            ),

  on "Stop"<a_timer_operation>:ATimerStop(
            <id>,
            <optional_if_predicate>,
            <reqmts_trace>
            ),

  on "Start"<a_timer_operation>:ATimerStart(
            <id>,
            <optional_if_predicate>,
            <reqmts_trace>
            )
  ;

transform optional_if_predicate
  on "Expression"
      <optional_if_predicate>:OptIfPredicate(
            <c_expression>
            )
  ;

/*transform if_expression
  on "IfExpression"
      <if_expression>:IfExpression(
            <c_expression>
            )
  ;
*/

transform c_expression
  on "Identifier"<c_expression>: CIdentifier(<id>),
  on "TypeExpresion"
      <c_expression>: CTypeExpression(
            <type_name>,
            <id>,
/*
            <c_expression_list>
*/
            <c_initial_args>
                ),

  on "Textual"<c_expression>:
       CTextual_Description(<commentLines>),
/*
  on "Textual"<c_expression>:
       CTextual_Description(<string_lit>),
*/

  on "Time"<c_expression>: CTimeExpression(<time>),
```

APPENDIX F - Transformation Rules

```
on "(<c_expression>)"
    <c_expression>: CParenthesizedExp(<c_expression>),

/* c_boolean_expression */

on "True"<c_expression>: CTrue(),

on "False"<c_expression>: CFalse(),

on "Not"<c_expression>: CNotExp(
          <c_expression>
          ),

on "="<c_expression>: CEqualExp(
          <c_expression>,
          <c_expression>
          ),

on "<"<c_expression>: CLessExp(
          <c_expression>,
          <c_expression>
          ),

on ">"<c_expression>: CGreaterExp(
          <c_expression>,
          <c_expression>
          ),

on ">="<c_expression>: CGreatEqualExp(
          <c_expression>,
          <c_expression>
          ),

on "<="<c_expression>: CLessEqualExp(
          <c_expression>,
          <c_expression>
          ),

on "/="<c_expression>: CNotEquivExp(
          <c_expression>,
          <c_expression>
          ),

on "And"<c_expression>: CAndExp(
          <c_expression>,
          <c_expression>
          ),

on "Or"<c_expression>: COrExp(
          <c_expression>,
          <c_expression>
          ),

on "Xor"<c_expression>: CXorExp(
          <c_expression>,
          <c_expression>
          ),

/* c_arithmetic_expression */

on "Integer"<c_expression>: CInteger(<integer>),

on "Real"<c_expression>: CReal
          (<integer>, <integer>),

  on "+"<c_expression>: CPlusExp(
          <c_expression>,
          <c_expression>
          ),

  on "-"<c_expression>: CMinusExp(
          <c_expression>,
          <c_expression>
          ),

  on "*"<c_expression>: CTimesExp(
          <c_expression>,
          <c_expression>
          ),

  on "/"<c_expression>: CDivExp(
          <c_expression>,
          <c_expression>
          ),

  on "Negative"<c_expression>: CNegativeExp(
          <c_expression>
          ),

  on "Mod"<c_expression>: CModExp(
          <c_expression>,
          <c_expression>
          ),

  on "Rem"<c_expression>: CRemExp(
          <c_expression>,
          <c_expression>
          ),

  on "**"<c_expression>: CExponentExp(
          <c_expression>,
          <c_expression>
          ),

  /* string expressions */
  on "&"<c_expression>: CConcatExp(
          <c_expression>,
          <c_expression>
          )
  ;

transform c_an_argument
  on "C_Exp_Argument"
     <c_an_argument>: CAnArgument(
        <c_expression_list>
          )
  ;
```

```
PROTOTYPE(synthesized prototype t;);

PSDL_COMPONENTS(
        synthesized psdl_components reversed;
      );

COMPONENT(synthesized component t;);
ID    (synthesized id t;);
INTEGER(synthesized integer t;);
TYPE_SPEC(synthesized type_spec t;);
O_GENERIC_PARAMS(synthesized o_generic_params t;);


TYPE_DECLARATIONS
      (
        inherited type_declarations tail;
        synthesized type_declarations reversed;
      );


A_DECL (synthesized a_decl t;);
TYPE_NAME(synthesized type_name t;);
DECL_TYPE_NAME(synthesized decl_type_name t;);

BRACKET_TYPE_DECLARATIONS
      (synthesized bracket_type_declarations t;);

O_BRACKET_TYPE_DECLARATIONS
      (synthesized o_bracket_type_declarations t;);

ALONE_ID_LIST(
            inherited alone_id_list tail;
            synthesized alone_id_list reversed;
            );

ID_LIST  (
            inherited id_list tail;
            synthesized id_list reversed;
            );


O_TYPE_DECLS(synthesized o_type_decls t;);

O_OPERATORS(
            inherited o_operators tail;
            synthesized o_operators reversed;
            );

T_OPER_SPEC(synthesized t_oper_spec t;);
OPERATOR_SPEC(synthesized operator_spec t;);

O_GENERICS_LIST(
            inherited o_generics_list tail;
            synthesized o_generics_list reversed;
            );

O_GENERICS(synthesized o_generics t;);

O_INPUTS_LIST(
            inherited o_inputs_list tail;
            synthesized o_inputs_list reversed;
            );
```

```
O_INPUTS(synthesized o_inputs t;);

O_OUTPUTS_LIST(
            inherited o_outputs_list tail;
            synthesized o_outputs_list reversed;
            );

O_OUTPUTS(synthesized o_outputs t;);

O_EXCEPTIONS_LIST(
            inherited o_exceptions_list tail;
            synthesized o_exceptions_list reversed;
            );

O_EXCEPTIONS(synthesized o_exceptions t;);

O_TIMING_INFO(synthesized o_timing_info t;);
TIME_UNIT(synthesized time_unit t;);
TIME  (synthesized time t;);

O_STATES_LIST(
      inherited o_states_list tail;
      synthesized o_states_list reversed;
      );

O_STATES(synthesized o_states t;);


INITIAL_ARGS(
      inherited initial_args tail;
      synthesized initial_args reversed;
      );

AN_ARGUMENT(synthesized an_argument  t;);

C_INITIAL_ARGS(
      inherited c_initial_args tail;
      synthesized c_initial_args reversed;
      );

C_AN_ARGUMENT(synthesized c_an_argument  t;);

EXPRESSION_LIST(
      inherited expression_list tail;
      synthesized expression_list reversed;
      );

EXPRESSION(synthesized expression t;);

C_EXPRESSION_LIST(
      inherited c_expression_list tail;
      synthesized c_expression_list reversed;
      );

C_EXPRESSION(synthesized c_expression t;);

O_KEYWORDS(synthesized o_keywords t;);
O_INFORMAL_DESCS(synthesized o_informal_descs t;);
O_FORMAL_DESCS(synthesized o_formal_descs t;);
REQMTS_TRACE(synthesized reqmts_trace t;);

/*NEW CONCRETE INPUT SYMBOLS
*/
```

```
TYPE_IMPL(synthesized type_impl  t;);
OPERATOR_IMPL(synthesized operator_impl  t;);

OPERATOR_IMPL_LIST(
            inherited operator_impl_list tail;
            synthesized operator_impl_list  reversed;
            );

GRAPH    (synthesized graph  t;);
DECLARATIONS(synthesized declarations  t;);
CC           (synthesized cc  t;);
T_OP_IMPL(synthesized t_op_impl  t;);
VERTEX_LIST(
            inherited vertex_list tail;
            synthesized vertex_list reversed;
            );

A_VERTEX(synthesized a_vertex  t;);
OPERATOR_ID(synthesized operator_id  t;);

OPTIONAL_TYPE_ID(synthesized optional_type_id t;);

OPERATOR_ID_PAIRS(synthesized operator_id_pairs  t;);
OPTIONAL_TIME(synthesized optional_time  t;);
LATENCY_TIME(synthesized latency_time t;);
EDGE_LIST(
            inherited edge_list tail;
            synthesized edge_list reversed;
            );
AN_EDGE  (synthesized an_edge  t;);
FROM_VERTEX_ID(synthesized from_vertex_id t;);
TO_VERTEX_ID(synthesized to_vertex_id t;);
OPTIONAL_STREAMS(synthesized optional_streams  t;);
OPTIONAL_TIMERS(synthesized optional_timers  t;);

CONSTRAINTS(inherited constraints  tail;
             synthesized constraints  reversed;);

A_CONSTRAINT(synthesized a_constraint  t;);
OPTIONAL_TRIGGER(synthesized optional_trigger  t;);
TYPE_OF_TRIGGER(synthesized type_of_trigger t;);
OPTIONAL_PERIOD(synthesized optional_period  t;);
OPTIONAL_FINISH_WITHIN(synthesized optional_finish_within  t;);
OPTIONAL_MCP(synthesized optional_mcp  t;);
OPTIONAL_MRT(synthesized optional_mrt  t;);

OUTPUT_GUARDS(
       inherited output_guards  tail;
       synthesized output_guards  reversed;
      );

A_GUARD(synthesized a_guard  t;);
EXCEPTION_OPS(synthesized exception_ops  t;);

EXCEPTION_OPTIONS(
       inherited exception_options  tail;
       synthesized exception_options  reversed;
      );

AN_EXCEPTION(synthesized an_exception  t;);

TIMER_OPERATIONS(inherited timer_operations  tail;
       synthesized timer_operations  reversed;);

A_TIMER_OPERATION(synthesized a_timer_operation  t;);
OPTIONAL_IF_PREDICATE(synthesized optional_if_predicate  t;);


prototype~ PROTOTYPE.t;
psdl_components~ PSDL_COMPONENTS.reversed;

component~ COMPONENT.t;
id       ~ ID.t;
integer~ INTEGER.t;
type_spec~ TYPE_SPEC.t;
o_generic_params~ O_GENERIC_PARAMS.t;

type_declarations~ TYPE_DECLARATIONS.reversed
        (TYPE_DECLARATIONS.tail=TypeDeclNil;);

a_decl ~ A_DECL.t;
type_name~ TYPE_NAME.t;
decl_type_name~ DECL_TYPE_NAME.t;
bracket_type_declarations
      ~ BRACKET_TYPE_DECLARATIONS.t;

o_bracket_type_declarations
      ~ O_BRACKET_TYPE_DECLARATIONS.t;


alone_id_list~ ALONE_ID_LIST.reversed
                (ALONE_ID_LIST.tail=AIdNil;);

id_list  ~ ID_LIST.reversed
                (ID_LIST.tail=IdNil;);

o_type_decls~ O_TYPE_DECLS.t;

o_operators~ O_OPERATORS.reversed
                (O_OPERATORS.tail=OperatorNil;);

t_oper_spec~ T_OPER_SPEC.t;
operator_spec~ OPERATOR_SPEC.t;
o_generics_list~ O_GENERICS_LIST.reversed
                (O_GENERICS_LIST.tail=GenericsListNone;);

o_generics~ O_GENERICS.t;
o_inputs_list~ O_INPUTS_LIST.reversed
                (O_INPUTS_LIST.tail=InputsListNone;);

o_inputs~ O_INPUTS.t;

o_outputs_list~ O_OUTPUTS_LIST.reversed
                (O_OUTPUTS_LIST.tail=OutputsListNone;);

o_outputs~ O_OUTPUTS.t;
o_exceptions_list~ O_EXCEPTIONS_LIST.reversed
                (O_EXCEPTIONS_LIST.tail=ExcListNone;);

o_exceptions~ O_EXCEPTIONS.t;

o_timing_info~ O_TIMING_INFO.t;
time_unit~ TIME_UNIT.t;
time   ~ TIME.t;
```

```
o_states_list~ O_STATES_LIST.reversed
        {O_STATES_LIST.tail=StatesListNone;};

o_states~ O_STATES.t;

an_argument~ AN_ARGUMENT.t;
c_an_argument~ C_AN_ARGUMENT.t;

initial_args~ INITIAL_ARGS.reversed
        {INITIAL_ARGS.tail=InitialArgsNil;};

c_initial_args~ C_INITIAL_ARGS.reversed
        {C_INITIAL_ARGS.tail=CInitialArgsNil;};

expression_list~ EXPRESSION_LIST.reversed
            {EXPRESSION_LIST.tail=InitialExpListNil;};

expression~ EXPRESSION.t;

c_expression_list~ C_EXPRESSION_LIST.reversed
            {C_EXPRESSION_LIST.tail=CInitialExpListNil;};
c_expression~ C_EXPRESSION.t;

o_keywords~ O_KEYWORDS.t;
o_informal_descs~ O_INFORMAL_DESCS.t;
o_formal_descs~ O_FORMAL_DESCS.t;
reqmts_trace~ REQMTS_TRACE.t;
type_impl~ TYPE_IMPL.t;
operator_impl~ OPERATOR_IMPL.t;

operator_impl_list      ~ OPERATOR_IMPL_LIST.reversed
          {OPERATOR_IMPL_LIST.tail=OpImplListNull;};

graph                   ~ GRAPH.t;
declarations~ DECLARATIONS.t;
cc      ~ CC.t;
t_op_impl               ~ T_OP_IMPL.t;

vertex_list~ VERTEX_LIST.reversed
        {VERTEX_LIST.tail=VertexListNull;};

a_vertex                ~ A_VERTEX.t;
operator_id             ~ OPERATOR_ID.t;
optional_type_id        ~ OPTIONAL_TYPE_ID.t;
operator_id_pairs       ~ OPERATOR_ID_PAIRS.t;
optional_time           ~ OPTIONAL_TIME.t;
latency_time~ LATENCY_TIME.t;

edge_list               ~ EDGE_LIST.reversed
                {EDGE_LIST.tail=EdgeListNil;};

an_edge                 ~ AN_EDGE.t;
from_vertex_id~ FROM_VERTEX_ID.t;
to_vertex_id~ TO_VERTEX_ID.t;
optional_streams        ~ OPTIONAL_STREAMS.t;
optional_timers         ~ OPTIONAL_TIMERS.t;

constraints             ~ CONSTRAINTS.reversed
        {CONSTRAINTS.tail=ConstraintsNull;};

a_constraint            ~ A_CONSTRAINT.t;
optional_trigger        ~ OPTIONAL_TRIGGER.t;
type_of_trigger~ TYPE_OF_TRIGGER.t;
```

```
optional_period         ~ OPTIONAL_PERIOD.t;
optional_finish_within  ~ OPTIONAL_FINISH_WITHIN.t;
optional_mcp~ OPTIONAL_MCP.t;
optional_mrt            ~ OPTIONAL_MRT.t;

output_guards           ~ OUTPUT_GUARDS.reversed
        {OUTPUT_GUARDS.tail=OutputGuardsNil;};

a_guard~ A_GUARD.t;
exception_ops           ~ EXCEPTION_OPS.t;

exception_options       ~ EXCEPTION_OPTIONS.reversed
          {EXCEPTION_OPTIONS.tail=
           ExceptionOptionsNil;};

timer_operations        ~ TIMER_OPERATIONS.reversed
        {TIMER_OPERATIONS.tail=
            TimerOperationsNil;};

a_timer_operation       ~ A_TIMER_OPERATION.t;
optional_if_predicate   ~ OPTIONAL_IF_PREDICATE.t;


/* ==================== PRECEDENCE DECLARATIONS ==================== */


left ANDKW ORKW XORKW;
left '<' '>' '=' GTEKW LTEKW NEQVKW;
left '+' '-' '&';
left '*' '/' MODKW REMKW;
left EXPKW ABSKW NOTKW;


/*=================================================================*/


/*  CONCRETE GRAMMAR'S PRODUCTIONS
*/
PROTOTYPE
  ::= (PSDL_COMPONENTS)
    {
     PROTOTYPE.t=Prot(PSDL_COMPONENTS.reversed);
    }
  ;

PSDL_COMPONENTS
  ::= (){PSDL_COMPONENTS.reversed = PsdlNil;}
  |   (COMPONENT PSDL_COMPONENTS){
      PSDL_COMPONENTS$1.reversed=
            COMPONENT.t::PSDL_COMPONENTS$2.reversed;
      }
  ;

COMPONENT
  ::= (TYPEKW ID TYPE_SPEC TYPE_IMPL)
    {COMPONENT.t = (Data(
              ID.t,
              TYPE_SPEC.t,
              TYPE_IMPL.t
             )
        );}
```

```
  |   (OPERATORKW ID OPERATOR_SPEC OPERATOR_IMPL)
    (COMPONENT.t = (Op(
           ID.t,
           OPERATOR_SPEC.t,
           OPERATOR_IMPL.t
          )
             );)
  ;


ID::= (IDENTIFIER)
    (ID.t=Id(IDENTIFIER);)
  ;

INTEGER ::= (INTEGERS)
    (INTEGER.t=IntegerVal(INTEGERS);)
  ;

TYPE_SPEC
  ::= (SPECKW
       O_GENERIC_PARAMS
       O_TYPE_DECLS
       O_OPERATORS
       O_KEYWORDS
       O_INFORMAL_DESCS
       O_FORMAL_DESCS ENDKW)
    (O_OPERATORS.tail=OperatorNil;
     TYPE_SPEC.t=TypeSpec(
        O_GENERIC_PARAMS.t,
        O_TYPE_DECLS.t,
        O_OPERATORS.reversed,
        O_KEYWORDS.t,
        O_INFORMAL_DESCS.t,
        O_FORMAL_DESCS.t
             );)
  ;


O_GENERIC_PARAMS
  ::= ()(O_GENERIC_PARAMS.t=(GenericNone);)

  |   (GENERICKW TYPE_DECLARATIONS)
    (TYPE_DECLARATIONS.tail=TypeDeclNil;
    O_GENERIC_PARAMS.t=Generic(
             TYPE_DECLARATIONS.reversed
                );)
  ;

TYPE_DECLARATIONS
  ::= (A_DECL)
    (TYPE_DECLARATIONS.reversed=(
      A_DECL.t::TYPE_DECLARATIONS.tail
                    );)

  |   (TYPE_DECLARATIONS ',' A_DECL)
    (TYPE_DECLARATIONS$2.tail=(
      A_DECL.t::TYPE_DECLARATIONS$1.tail
              );

     TYPE_DECLARATIONS$1.reversed=
      TYPE_DECLARATIONS$2.reversed;
    )
  ;


A_DECL::= (ID_LIST ':' DECL_TYPE_NAME)
    (ID_LIST.tail = IdNil;
     A_DECL.t = ADecl(ID_LIST.reversed, DECL_TYPE_NAME.t);
    )
  ;

TYPE_NAME
  ::= (ID O_BRACKET_TYPE_DECLARATIONS)
    (TYPE_NAME.t = TypeName(
         ID.t,
         O_BRACKET_TYPE_DECLARATIONS.t
                );)
  ;

DECL_TYPE_NAME
  ::= (INTKW)
    (DECL_TYPE_NAME.t = DTypeInteger();)

  |   (REALKW)
    (DECL_TYPE_NAME.t = DTypeReal();)

  |   (BOOLKW)
    (DECL_TYPE_NAME.t = DTypeBoolean();)
  |   (ID)
    (DECL_TYPE_NAME.t = DTypeSimpleId(ID.t);)

  |   (ID BRACKET_TYPE_DECLARATIONS)
    (DECL_TYPE_NAME.t = DTypeUserDefined(
             ID.t,
             BRACKET_TYPE_DECLARATIONS.t);)
  ;


BRACKET_TYPE_DECLARATIONS
  ::= ('[' TYPE_DECLARATIONS ']')
    (TYPE_DECLARATIONS.tail=TypeDeclNil;
     BRACKET_TYPE_DECLARATIONS.t =
        BTypeDeclaration(
             TYPE_DECLARATIONS.reversed
             );)
  ;


O_BRACKET_TYPE_DECLARATIONS
  ::= ()(O_BRACKET_TYPE_DECLARATIONS.t = OBTypeNone();)
  |   ('[' TYPE_DECLARATIONS ']')
    (TYPE_DECLARATIONS.tail=TypeDeclNil;
     O_BRACKET_TYPE_DECLARATIONS.t =
        OBTypeDeclaration(
             TYPE_DECLARATIONS.reversed
             );)
  ;

ALONE_ID_LIST
  ::= ()(ALONE_ID_LIST$1.reversed = AIdNil;)

  |   (ID)
    (ALONE_ID_LIST$1.reversed=(ID.t::ALONE_ID_LIST$1.tail);)

  |   (ALONE_ID_LIST ',' ID)(
      ALONE_ID_LIST$2.tail=(ID.t::ALONE_ID_LIST$1.tail);
      ALONE_ID_LIST$1.reversed=ALONE_ID_LIST$2.reversed;
```

```
    }
  ;

ID_LIST::= (ID)
    (ID_LIST$1.reversed=(ID.t::ID_LIST$1.tail);)

  |   (ID_LIST ',' ID){
    ID_LIST$2.tail=(ID.t::ID_LIST$1.tail);
    ID_LIST$1.reversed=ID_LIST$2.reversed;
    }
  ;

O_TYPE_DECLS
  ::= ()(O_TYPE_DECLS.t=(TypeNone);)

  |   (TYPE_DECLARATIONS)
    (TYPE_DECLARATIONS.tail=TypeDeclNil;
     O_TYPE_DECLS.t=Type(TYPE_DECLARATIONS.reversed);)
  ;

O_OPERATORS
  ::= ()(O_OPERATORS.reversed=(O_OPERATORS.tail);)

  |   (O_OPERATORS T_OPER_SPEC)
    (O_OPERATORS$2.tail=(
        T_OPER_SPEC.t::
        O_OPERATORS$1.tail);

     O_OPERATORS$1.reversed=O_OPERATORS$2.reversed;
    )
  ;

T_OPER_SPEC
  ::= (OPERATORKW ID OPERATOR_SPEC)
    (T_OPER_SPEC.t = TOpSpec(
        ID.t,
        OPERATOR_SPEC.t
            );)
  ;

OPERATOR_SPEC
  ::= (SPECKW
       O_GENERICS_LIST
       O_INPUTS_LIST
       O_OUTPUTS_LIST
       O_STATES_LIST
       O_EXCEPTIONS_LIST
       O_TIMING_INFO
       O_KEYWORDS
       O_INFORMAL_DESCS
       O_FORMAL_DESCS
       ENDKW)
    (O_GENERICS_LIST.tail=GenericsListNone;
     O_INPUTS_LIST.tail=InputsListNone;
     O_OUTPUTS_LIST.tail=OutputsListNone;
     O_STATES_LIST.tail=StatesListNone;
     O_EXCEPTIONS_LIST.tail=ExcListNone;
     OPERATOR_SPEC.t = OperatorSpec(
            O_GENERICS_LIST.reversed,
            O_INPUTS_LIST.reversed,
            O_OUTPUTS_LIST.reversed,
            O_STATES_LIST.reversed,
            O_EXCEPTIONS_LIST.reversed,
            O_TIMING_INFO.t,
            O_KEYWORDS.t,
            O_INFORMAL_DESCS.t,
            O_FORMAL_DESCS.t
                );)
  ;

O_GENERICS_LIST
  ::= ()(O_GENERICS_LIST.reversed = (O_GENERICS_LIST.tail);)

  |   (O_GENERICS_LIST O_GENERICS)
    (O_GENERICS_LIST$2.tail=(
        O_GENERICS.t::
        O_GENERICS_LIST$1.tail);

     O_GENERICS_LIST$1.reversed=O_GENERICS_LIST$2.reversed;
    )
  ;


O_GENERICS
  ::=   (GENERICKW TYPE_DECLARATIONS REQMTS_TRACE)
    (TYPE_DECLARATIONS.tail=TypeDeclNil;
     O_GENERICS.t = OpGenerics(
            TYPE_DECLARATIONS.reversed,
            REQMTS_TRACE.t
              );)
  ;

O_INPUTS_LIST
  ::= ()(O_INPUTS_LIST.reversed = (O_INPUTS_LIST.tail);)

  |   (O_INPUTS_LIST O_INPUTS)
    (O_INPUTS_LIST$2.tail=(
        O_INPUTS.t::
        O_INPUTS_LIST$1.tail);

     O_INPUTS_LIST$1.reversed=O_INPUTS_LIST$2.reversed;
    )
  ;

O_INPUTS
  ::= (INPUTKW TYPE_DECLARATIONS REQMTS_TRACE)
    (TYPE_DECLARATIONS.tail=TypeDeclNil;
     O_INPUTS.t = OpInputs(
            TYPE_DECLARATIONS.reversed,
            REQMTS_TRACE.t
              );)
  ;

O_OUTPUTS_LIST
  ::= ()(O_OUTPUTS_LIST.reversed = (O_OUTPUTS_LIST.tail);).

  |   (O_OUTPUTS_LIST O_OUTPUTS)
    (O_OUTPUTS_LIST$2.tail=(
        O_OUTPUTS.t::
        O_OUTPUTS_LIST$1.tail);

     O_OUTPUTS_LIST$1.reversed=O_OUTPUTS_LIST$2.reversed;
    )
  ;
```

```
O_OUTPUTS
  ::=   (OUTPUTKW TYPE_DECLARATIONS REQMTS_TRACE)
    {TYPE_DECLARATIONS.tail=TypeDeclNil;
     O_OUTPUTS.t = OpOutputs(
             TYPE_DECLARATIONS.reversed,
             REQMTS_TRACE.t
                 );}
  ;

O_EXCEPTIONS_LIST
  ::= (){O_EXCEPTIONS_LIST.reversed = (O_EXCEPTIONS_LIST.tail);}

    {O_EXCEPTIONS_LIST$2.tail=(
        O_EXCEPTIONS.t::
        O_EXCEPTIONS_LIST$1.tail);

     O_EXCEPTIONS_LIST$1.reversed=O_EXCEPTIONS_LIST$2.reversed;
    }
  ;

O_EXCEPTIONS
  ::=(EXCEPTIONSKW ALONE_ID_LIST REQMTS_TRACE)
    {ALONE_ID_LIST.tail = AIdNil;
     O_EXCEPTIONS.t = OpExceptions(
             ALONE_ID_LIST.reversed,
             REQMTS_TRACE.t
                     );}
  ;

O_TIMING_INFO
  ::=()
    {O_TIMING_INFO.t = (OpTimingInfoNone);}

  | (MAXEXTIMEKW TIME REQMTS_TRACE)
    {O_TIMING_INFO.t = OpTimingInfo(
             TIME.t,
             REQMTS_TRACE.t
                      );}
  ;

TIME::= (INTEGER TIME_UNIT)
    {TIME.t = Time(INTEGER.t, TIME_UNIT.t);}
  ;

TIME_UNIT
  ::= (MICRO)
    {TIME_UNIT.t = UnitMICROSECONDS;}
  |   (MS)
    {TIME_UNIT.t = UnitMS;}
  |   (SEC)
    {TIME_UNIT.t = UnitSEC;}
  |   (MIN)
    {TIME_UNIT.t = UnitMIN;}
  |   (HOURS)
    {TIME_UNIT.t = UnitHOURS;}
  ;

O_STATES_LIST
  ::= (){O_STATES_LIST.reversed = (O_STATES_LIST.tail);}

  |   (O_STATES_LIST O_STATES)
    {O_STATES_LIST$2.tail=(
        O_STATES.t::
        O_STATES_LIST$1.tail);

     O_STATES_LIST$1.reversed=O_STATES_LIST$2.reversed;
    }
  ;

O_STATES
  ::=   (STATESKW TYPE_DECLARATIONS INITIALKW EXPRESSION_LIST REQMTS_TRACE)
    {EXPRESSION_LIST.tail=InitialExpListNil;
     TYPE_DECLARATIONS.tail=TypeDeclNil;
     O_STATES.t = OpStates(
         TYPE_DECLARATIONS.reversed,
         EXPRESSION_LIST.reversed,
         REQMTS_TRACE.t
               );}
  ;

AN_ARGUMENT
  ::= ('(' EXPRESSION_LIST ')')
    {EXPRESSION_LIST.tail = InitialExpListNil;
     AN_ARGUMENT.t=AnArgument(EXPRESSION_LIST.reversed);
    }
  ;

INITIAL_ARGS
  ::= (){INITIAL_ARGS.reversed=(INITIAL_ARGS.tail);}

  |   (INITIAL_ARGS AN_ARGUMENT)
    {INITIAL_ARGS$2.tail=
      (AN_ARGUMENT.t::INITIAL_ARGS$1.tail);
     INITIAL_ARGS$1.reversed=INITIAL_ARGS$2.reversed;
    }
  ;

EXPRESSION_LIST
  ::= (EXPRESSION)
    {EXPRESSION_LIST.reversed=
      (EXPRESSION.t::EXPRESSION_LIST.tail);}

  |   (EXPRESSION_LIST ',' EXPRESSION)
    {EXPRESSION_LIST$2.tail=
      (EXPRESSION.t::EXPRESSION_LIST$1.tail);
     EXPRESSION_LIST$1.reversed=EXPRESSION_LIST$2.reversed;
    }
  ;

EXPRESSION
  ::= (ID)
    {EXPRESSION.t=Identifier(ID.t);}

  |   (QUOTEKW yCommentLines QUOTEKW)
    { EXPRESSION.t = Textual_Description(yCommentLines.a); }

  |  (TYPE_NAME '.' ID  INITIAL_ARGS )
    {INITIAL_ARGS.tail = InitialArgsNil;

     EXPRESSION.t = TypeExpression(
             TYPE_NAME.t,
             ID.t,
             INITIAL_ARGS.reversed
                   );}

  |  ('('EXPRESSION')')
```

```
  (EXPRESSION$1.t=ParenthesizedExp(EXPRESSION$2.t);)

/* BOOLEAN_EXPRESSION */
| (NOTKW EXPRESSION prec NOTKW)
  (EXPRESSION$1.t=
    NotExp(EXPRESSION$2.t);)
|   (FALSEKW)
  (EXPRESSION.t=False;)

|   (TRUEKW)
  (EXPRESSION.t=True;)

|   (EXPRESSION '=' EXPRESSION)
  (EXPRESSION$1.t=
    EqualExp(
      EXPRESSION$2.t,
      EXPRESSION$3.t
      );)

|   (EXPRESSION '<' EXPRESSION prec '<')
  (EXPRESSION$1.t=
    LessExp(
      EXPRESSION$2.t,
      EXPRESSION$3.t
          );)

|   (EXPRESSION '>' EXPRESSION prec '>')
  (EXPRESSION$1.t=
    GreaterExp(
      EXPRESSION$2.t,
      EXPRESSION$3.t
        );)

|   (EXPRESSION GTEKW EXPRESSION prec GTEKW)
  (EXPRESSION$1.t=
    GreatEqualExp(
      EXPRESSION$2.t,
      EXPRESSION$3.t
          );)

|   (EXPRESSION LTEKW EXPRESSION prec LTEKW)
  (EXPRESSION$1.t=
    LessEqualExp(
      EXPRESSION$2.t,
      EXPRESSION$3.t
        );)

|   (EXPRESSION NEQVKW EXPRESSION prec NEQVKW)
  (EXPRESSION$1.t=
    NotEquivExp(
      EXPRESSION$2.t,
      EXPRESSION$3.t
        );)

|   (EXPRESSION  ANDKW  EXPRESSION prec ANDKW)
  (EXPRESSION$1.t=
    AndExp(
      EXPRESSION$2.t,
      EXPRESSION$3.t
        );)

|   (EXPRESSION ORKW EXPRESSION prec ORKW)
  (EXPRESSION$1.t=
    OrExp(
      EXPRESSION$2.t,
      EXPRESSION$3.t
        );)

|   (EXPRESSION XORKW EXPRESSION prec XORKW)
  (EXPRESSION$1.t=
    XorExp(
      EXPRESSION$2.t,
      EXPRESSION$3.t);)

/* ARITHMETIC_EXPRESSION */
|   (INTEGER)
  (EXPRESSION.t=Integer(INTEGER.t);)

|   (INTEGER '.' INTEGER)
  (EXPRESSION.t=Real(
          INTEGER$1.t,
          INTEGER$2.t
              );)

|   (EXPRESSION '+' EXPRESSION prec '+')
  (EXPRESSION$1.t=
    PlusExp(
      EXPRESSION$2.t,
      EXPRESSION$3.t
          );)

|   (EXPRESSION '-' EXPRESSION prec '-')
  (EXPRESSION$1.t=
    MinusExp(
      EXPRESSION$2.t,
      EXPRESSION$3.t
      );)

|   (EXPRESSION '*' EXPRESSION prec '*')
  (EXPRESSION$1.t=
    TimesExp(
      EXPRESSION$2.t,
      EXPRESSION$3.t
      );)

|   (EXPRESSION '/' EXPRESSION prec '/')
  (EXPRESSION$1.t=
    DivExp(
      EXPRESSION$2.t,
      EXPRESSION$3.t
        );)

|   ('-' EXPRESSION prec '-')
  (EXPRESSION$1.t=
    NegativeExp(
      EXPRESSION$2.t
        );)

|   ('+' EXPRESSION prec '+')
  (EXPRESSION$1.t=
    PositiveExp(
      EXPRESSION$2.t
        );)

|   (ABSKW '('EXPRESSION')' prec ABSKW)
  (EXPRESSION$1.t=
```

```
      AbsExp(
        EXPRESSION$2.t
            );)

  |   (EXPRESSION MODKW EXPRESSION prec MODKW)
    (EXPRESSION$1.t=
      ModExp(
        EXPRESSION$2.t,
        EXPRESSION$3.t
            );)

  |   (EXPRESSION REMKW EXPRESSION prec REMKW)
    (EXPRESSION$1.t=
      RemExp(
        EXPRESSION$2.t,
        EXPRESSION$3.t
            );)

  |   (EXPRESSION EXPKW EXPRESSION prec EXPKW)
    (EXPRESSION$1.t=
      ExponentExp(
        EXPRESSION$2.t,
        EXPRESSION$3.t
           );)

  /* string expression */

  |   (EXPRESSION '&' EXPRESSION prec '&')
    (EXPRESSION$1.t=
      ConcatExp(
        EXPRESSION$2.t,
        EXPRESSION$3.t
            );)
  ;

O_KEYWORDS
  ::= ()(O_KEYWORDS.t=(KeyWordsNone);)

  |   (KEYWKW ALONE_ID_LIST)
    (ALONE_ID_LIST.tail=AIdNil;
     O_KEYWORDS.t=KeyWords(ALONE_ID_LIST.reversed);)
  ;

O_INFORMAL_DESCS
  ::= ()(O_INFORMAL_DESCS.t=(InformalDescsNull);)

  |   (DESCRIPKW LCURLY yCommentLines RCURLY)
    { O_INFORMAL_DESCS.t = InformalDescs(yCommentLines.a); }
  ;

O_FORMAL_DESCS
  ::= ()(O_FORMAL_DESCS.t=(FormalDescsNone);)

  |   (AXIOMKW LCURLY yCommentLines RCURLY)
    { O_FORMAL_DESCS.t=FormalDescs(yCommentLines.a); }
  ;

REQMTS_TRACE
  ::= () (REQMTS_TRACE.t=(ReqmtsTraceNone);)

  |   (REQBYKW ALONE_ID_LIST)
      (ALONE_ID_LIST.tail=AIdNil;
       REQMTS_TRACE.t=ReqmtsTrace(ALONE_ID_LIST.reversed);)
  ;

TYPE_IMPL
  ::= () ($$.t = (TypeImplNull);)

  |   (IMPLKW ADAKW ID ENDKW)
    (TYPE_IMPL.t=AdaTypeImpl(ID.t);)

  |   (IMPLKW TYPE_NAME OPERATOR_IMPL_LIST ENDKW)
    (OPERATOR_IMPL_LIST.tail=OpImplListNull;
     TYPE_IMPL.t=TypeImpl(
        TYPE_NAME.t,
        OPERATOR_IMPL_LIST.reversed
            );)
  ;

OPERATOR_IMPL
  ::= () ($$.t = (OpImplNull);)

  |   (IMPLKW ADAKW ID ENDKW)
    (OPERATOR_IMPL.t=AdaOpImpl(ID.t);)
  |   (IMPLKW GRAPH DECLARATIONS CC ENDKW)
    (OPERATOR_IMPL.t=OperatorImpl(
            GRAPH.t,
            DECLARATIONS.t,
            CC.t);)
  ;

OPERATOR_IMPL_LIST
  ::= ()(OPERATOR_IMPL_LIST.reversed=OPERATOR_IMPL_LIST.tail;)

  |  (OPERATOR_IMPL_LIST T_OP_IMPL)
    (OPERATOR_IMPL_LIST$2.tail=(
            T_OP_IMPL.t::
            OPERATOR_IMPL_LIST$1.tail);
     OPERATOR_IMPL_LIST$1.reversed=
            OPERATOR_IMPL_LIST$2.reversed;
    )
  ;

T_OP_IMPL
  ::= (OPERATORKW ID OPERATOR_IMPL)
    (T_OP_IMPL.t=TOpImpl(
            ID.t,
            OPERATOR_IMPL.t
              );)
  ;

GRAPH::= (GRAPHKW VERTEX_LIST EDGE_LIST)
    (EDGE_LIST.tail=EdgeListNil;
     VERTEX_LIST.tail=VertexListNull;
     GRAPH.t=Graph(
        VERTEX_LIST.reversed,
        EDGE_LIST.reversed
            );)
  ;

VERTEX_LIST
  ::= ()(VERTEX_LIST.reversed=VERTEX_LIST.tail;)
  |   (VERTEX_LIST A_VERTEX)
    (
     VERTEX_LIST$2.tail=(
        A_VERTEX.t::
```

```
        VERTEX_LIST$1.tail
            );
     VERTEX_LIST$1.reversed=VERTEX_LIST$2.reversed;
    }
  ;

A_VERTEX
  ::= (VERTEXKW OPERATOR_ID OPTIONAL_TIME)
    {A_VERTEX.t=AVertex(
        OPERATOR_ID.t,
        OPTIONAL_TIME.t
           );}
  ;

OPTIONAL_TYPE_ID
  ::= ()
    {OPTIONAL_TYPE_ID.t = OptionalTypeIdNull;}

  | (ID)
    {OPTIONAL_TYPE_ID.t = OptionalTypeId(ID.t);}
  ;

OPERATOR_ID
  ::= (ID OPERATOR_ID_PAIRS)
    {OPERATOR_ID.t=OperatorId(OptionalTypeIdNull,
                ID.t,
                OPERATOR_ID_PAIRS.t
                                              );}
  | (OPTIONAL_TYPE_ID '.' ID OPERATOR_ID_PAIRS)
    {OPERATOR_ID.t=OperatorId(OPTIONAL_TYPE_ID.t,
                ID.t,
                OPERATOR_ID_PAIRS.t
                                              );}
  ;

OPERATOR_ID_PAIRS
  ::= (){OPERATOR_ID_PAIRS.t=(OperatorIdPairsNull);}

  |   ('(' ALONE_ID_LIST '|' ALONE_ID_LIST ')')
    {ALONE_ID_LIST$1.tail = AIdNil;
     ALONE_ID_LIST$2.tail = AIdNil;
      OPERATOR_ID_PAIRS.t=OperatorIdPairs(
                ALONE_ID_LIST$1.reversed,
                ALONE_ID_LIST$2.reversed
                   );}
  ;
OPTIONAL_TIME
  ::= (){OPTIONAL_TIME.t=(OptionalTimeNull);}

  |   (':' TIME)
    {OPTIONAL_TIME.t=OptionalTime(
                TIME.t
                 );}
  ;

LATENCY_TIME
  ::= (){$$.t = (LatencyTimeNull);}

       |   (':' TIME)
              {LATENCY_TIME.t = LatencyTime(TIME.t);}
  ;

EDGE_LIST
  ::= (){EDGE_LIST.reversed=EDGE_LIST.tail;}
  |   (EDGE_LIST AN_EDGE)
    (
     EDGE_LIST$2.tail=(
        AN_EDGE.t::
        EDGE_LIST$1.tail
           );

     EDGE_LIST$1.reversed=EDGE_LIST$2.reversed;
    )
  ;

AN_EDGE ::= (EDGEKW ID LATENCY_TIME FROM_VERTEX_ID ARROWKW TO_VERTEX_ID)
    {AN_EDGE.t=AnEdge(
        ID.t,
        LATENCY_TIME.t,
        FROM_VERTEX_ID.t,
        TO_VERTEX_ID.t
         );}
  ;

FROM_VERTEX_ID
  ::=(ID OPERATOR_ID_PAIRS)
    {$$.t = FVertexId(OptionalTypeIdNull,
             ID.t,
             OPERATOR_ID_PAIRS.t
                                    );}

  | (OPTIONAL_TYPE_ID '.' ID OPERATOR_ID_PAIRS)
    {$$.t = FVertexId(
             OPTIONAL_TYPE_ID.t,
             ID.t,
             OPERATOR_ID_PAIRS.t
                                       );}
  ;

TO_VERTEX_ID
  ::=(ID OPERATOR_ID_PAIRS)
    {$$.t = TVertexId(OptionalTypeIdNull,
             ID.t,
             OPERATOR_ID_PAIRS.t
                                    );}

  | (OPTIONAL_TYPE_ID '.' ID OPERATOR_ID_PAIRS)
    {$$.t = TVertexId(
             OPTIONAL_TYPE_ID.t,
             ID.t,
             OPERATOR_ID_PAIRS.t
                                       );}
  ;


DECLARATIONS
  ::= (OPTIONAL_STREAMS OPTIONAL_TIMERS)
    {DECLARATIONS.t=Declarations(
             OPTIONAL_STREAMS.t,
             OPTIONAL_TIMERS.t
                 );}
  ;

OPTIONAL_STREAMS
  ::= (){OPTIONAL_STREAMS.t=(StreamsNull);}
```

```
  |   (DATSTRKW TYPE_DECLARATIONS)
    (TYPE_DECLARATIONS.tail=TypeDeclNil;
     OPTIONAL_STREAMS.t=Streams(
            TYPE_DECLARATIONS.reversed
               );)
  ;

OPTIONAL_TIMERS
  ::= ()(OPTIONAL_TIMERS.t=(TimersNull);)

  |   (TIMERKW ALONE_ID_LIST)
    (ALONE_ID_LIST.tail=AIdNil;
     OPTIONAL_TIMERS.t=Timers(
            ALONE_ID_LIST.reversed
             );)
  ;

CC::= () (CC.t=CcNull;)
  |   (CONTROLKW CONSTRAINTS O_INFORMAL_DESCS)
    (CONSTRAINTS.tail = ConstraintsNull;
     CC.t=Cc(
      CONSTRAINTS.reversed,
      O_INFORMAL_DESCS.t
      );)
  ;

CONSTRAINTS
  ::=  () (CONSTRAINTS.reversed=CONSTRAINTS.tail;)
  |   (CONSTRAINTS  A_CONSTRAINT)
    (CONSTRAINTS$2.tail=(
        A_CONSTRAINT.t::
        CONSTRAINTS$1.tail
            );
     CONSTRAINTS$1.reversed=CONSTRAINTS$2.reversed;
    )
  ;

A_CONSTRAINT
  ::= (OPERATORKW
       OPERATOR_ID
       OPTIONAL_TRIGGER
       OPTIONAL_PERIOD
       OPTIONAL_FINISH_WITHIN
       OPTIONAL_MCP
       OPTIONAL_MRT
       OUTPUT_GUARDS
       EXCEPTION_OPS
       TIMER_OPERATIONS)
    (TIMER_OPERATIONS.tail=TimerOperationsNil;
     OUTPUT_GUARDS.tail=OutputGuardsNil;
     A_CONSTRAINT.t=AConstraint(OPERATOR_ID.t,
                 OPTIONAL_TRIGGER.t,
                 OPTIONAL_PERIOD.t,
                 OPTIONAL_FINISH_WITHIN.t,
                 OPTIONAL_MCP.t,
                 OPTIONAL_MRT.t,
                 OUTPUT_GUARDS.reversed,
                 EXCEPTION_OPS.t,
                 TIMER_OPERATIONS.reversed
                )
    ;)
  ;
```

```
OPTIONAL_TRIGGER
  ::= ()(OPTIONAL_TRIGGER.t=(OptionalTriggerNull);)

  |   (TRIGGERBYKW
       TYPE_OF_TRIGGER
       ALONE_ID_LIST
       OPTIONAL_IF_PREDICATE
       REQMTS_TRACE)
    (ALONE_ID_LIST.tail=AIdNil;
     OPTIONAL_TRIGGER.t=OptionalTriggerAllOrSome(
                 TYPE_OF_TRIGGER.t,
                 ALONE_ID_LIST.reversed,
                 OPTIONAL_IF_PREDICATE.t,
                 REQMTS_TRACE.t
                    );)
  |   (TRIGGERKW IFKW
       EXPRESSION
       REQMTS_TRACE)
    (OPTIONAL_TRIGGER.t=OptionalIfExp(
      EXPRESSION.t,
      REQMTS_TRACE.t
      );)
  ;

TYPE_OF_TRIGGER
  ::= (ALLKW)(TYPE_OF_TRIGGER.t=TriggerAll();)
  |   (SOMEKW)(TYPE_OF_TRIGGER.t=TriggerSome();)
  ;

OPTIONAL_PERIOD
  ::= ()(OPTIONAL_PERIOD.t=(OptPeriodNull);)

  |   (PERKW TIME REQMTS_TRACE)
    (OPTIONAL_PERIOD.t=OptPeriod(
            TIME.t,
            REQMTS_TRACE.t
                );)
  ;

OPTIONAL_FINISH_WITHIN
  ::= ()(OPTIONAL_FINISH_WITHIN.t=(OptFinishWithinNull);)


  |   (FINISHKW TIME REQMTS_TRACE)
    (OPTIONAL_FINISH_WITHIN.t=OptFinishWithin(
                                  TIME.t,
                                  REQMTS_TRACE.t
                                   );)
  ;

OPTIONAL_MCP
  ::= ()(OPTIONAL_MCP.t=(OptMcpNull);)

  |   (MCPKW TIME REQMTS_TRACE)
    (OPTIONAL_MCP.t=OptMcp(
        TIME.t,
        REQMTS_TRACE.t
              );)
  ;

OPTIONAL_MRT
  ::= ()(OPTIONAL_MRT.t=(OptMrtNull);)
```

```
  |   (MRTKW TIME REQMTS_TRACE)
    {OPTIONAL_MRT.t=OptMrt(
        TIME.t,
        REQMTS_TRACE.t
              );}
  ;

OUTPUT_GUARDS
  ::= (){OUTPUT_GUARDS.reversed=OUTPUT_GUARDS.tail;}
  |   (OUTPUT_GUARDS A_GUARD)
    {OUTPUT_GUARDS$2.tail=(
        A_GUARD.t::
        OUTPUT_GUARDS$1.tail
              );

     OUTPUT_GUARDS$1.reversed=OUTPUT_GUARDS$2.reversed;
    }
  ;

A_GUARD::= (OUTPUTKW ALONE_ID_LIST IFKW C_EXPRESSION REQMTS_TRACE)
    {ALONE_ID_LIST.tail = AIdNil;
     A_GUARD.t=AGuard(
        ALONE_ID_LIST.reversed,
        C_EXPRESSION.t,
        REQMTS_TRACE.t
         );}
  ;

EXCEPTION_OPS
  ::= (){EXCEPTION_OPS.t=(ExceptionOpsNull);}

  |   (EXCEPTION_OPTIONS)
    {EXCEPTION_OPTIONS.tail=ExceptionOptionsNil;
     EXCEPTION_OPS.t=Exception(
            EXCEPTION_OPTIONS.reversed
              );}
  ;

EXCEPTION_OPTIONS
  ::= (AN_EXCEPTION)
    {EXCEPTION_OPTIONS.reversed=(
            AN_EXCEPTION.t::
            EXCEPTION_OPTIONS.tail
                );}

  |   (EXCEPTION_OPTIONS AN_EXCEPTION)
    {EXCEPTION_OPTIONS$2.tail=(
            AN_EXCEPTION.t::
            EXCEPTION_OPTIONS$1.tail
               );

     EXCEPTION_OPTIONS$1.reversed=EXCEPTION_OPTIONS$2.reversed;
    }
  ;

AN_EXCEPTION
  ::= (EXCEPTIONKW ID OPTIONAL_IF_PREDICATE REQMTS_TRACE)
    {AN_EXCEPTION.t=AnException(
            ID.t,
            OPTIONAL_IF_PREDICATE.t,
            REQMTS_TRACE.t
               );}
  ;
```

```
TIMER_OPERATIONS
  ::= (){TIMER_OPERATIONS.reversed=TIMER_OPERATIONS.tail;}
  |   (TIMER_OPERATIONS A_TIMER_OPERATION)
    {TIMER_OPERATIONS$2.tail=(
            A_TIMER_OPERATION.t::
            TIMER_OPERATIONS$1.tail
              );
     TIMER_OPERATIONS$1.reversed=TIMER_OPERATIONS$2.reversed;
    }
  ;

A_TIMER_OPERATION
  ::= (RESETKW ID OPTIONAL_IF_PREDICATE REQMTS_TRACE)
    {A_TIMER_OPERATION.t=ATimerReset(
                ID.t,
                OPTIONAL_IF_PREDICATE.t,
                REQMTS_TRACE.t
                );}

  |   (STOPKW ID OPTIONAL_IF_PREDICATE REQMTS_TRACE)
    {A_TIMER_OPERATION.t=ATimerStop(
                ID.t,
                OPTIONAL_IF_PREDICATE.t,
                REQMTS_TRACE.t
                );}

  |   (STARTKW ID OPTIONAL_IF_PREDICATE REQMTS_TRACE)
    {A_TIMER_OPERATION.t=ATimerStart(
                ID.t,
                OPTIONAL_IF_PREDICATE.t,
                REQMTS_TRACE.t
                );}
  ;

OPTIONAL_IF_PREDICATE
  ::= (){OPTIONAL_IF_PREDICATE.t=(OptIfPredicateNull);}

  |   (IFKW C_EXPRESSION)
    {OPTIONAL_IF_PREDICATE.t=OptIfPredicate(
                C_EXPRESSION.t
                      );}
  ;

C_AN_ARGUMENT
  ::= ('(' C_EXPRESSION_LIST ')')
    {C_EXPRESSION_LIST.tail = CInitialExpListNil;
     C_AN_ARGUMENT.t=CAnArgument(C_EXPRESSION_LIST.reversed);
    }
  ;

C_INITIAL_ARGS
  ::= (){C_INITIAL_ARGS.reversed=(C_INITIAL_ARGS.tail);}
  |   (C_INITIAL_ARGS C_AN_ARGUMENT)
    {C_INITIAL_ARGS$2.tail=
      (C_AN_ARGUMENT.t::C_INITIAL_ARGS$1.tail);
     C_INITIAL_ARGS$1.reversed=C_INITIAL_ARGS$2.reversed;
    }
  ;

C_EXPRESSION_LIST
  ::=   (C_EXPRESSION)
    {C_EXPRESSION_LIST.reversed=
```

```
      (C_EXPRESSION.t::C_EXPRESSION_LIST.tail);)

  |   (C_EXPRESSION_LIST ',' C_EXPRESSION)
    (C_EXPRESSION_LIST$2.tail=
      (C_EXPRESSION.t::C_EXPRESSION_LIST$1.tail);
     C_EXPRESSION_LIST$1.reversed=C_EXPRESSION_LIST$2.reversed;
    )
  ;

C_EXPRESSION
  ::=   (ID)
    (C_EXPRESSION.t=CIdentifier(ID.t);)

  |    ( QUOTEKW yCommentLines QUOTEKW)
    ( C_EXPRESSION.t= CTextual_Description(yCommentLines.a); )
  |   (TYPE_NAME '.' ID C_INITIAL_ARGS )
    (C_INITIAL_ARGS.tail = CInitialArgsNil;

     C_EXPRESSION.t = CTypeExpression(
            TYPE_NAME.t,
            ID.t,
            C_INITIAL_ARGS.reversed
                 );)

  |  ('('C_EXPRESSION')')
    (C_EXPRESSION$1.t=CParenthesizedExp(C_EXPRESSION$2.t);)

  |   (TIME)
    (C_EXPRESSION$1.t=CTimeExpression(TIME.t);)

  /* BOOLEAN_EXPRESSION */

  |   (NOTKW C_EXPRESSION prec NOTKW)
    (C_EXPRESSION$1.t=
      CNotExp(C_EXPRESSION$2.t);)
  |    (FALSEKW)
    (C_EXPRESSION.t=CFalse;)

  |    (TRUEKW)
    (C_EXPRESSION.t=CTrue;)

  |    (C_EXPRESSION '=' C_EXPRESSION prec '=')
    (C_EXPRESSION$1.t=
      CEqualExp(
        C_EXPRESSION$2.t,
        C_EXPRESSION$3.t
        );)

  |    (C_EXPRESSION '<' C_EXPRESSION prec '<')
    (C_EXPRESSION$1.t=
      CLessExp(
        C_EXPRESSION$2.t,
        C_EXPRESSION$3.t
            );)

  |    (C_EXPRESSION '>' C_EXPRESSION prec '>')
    (C_EXPRESSION$1.t=
      CGreaterExp(
        C_EXPRESSION$2.t,
        C_EXPRESSION$3.t
          );)

  |    (C_EXPRESSION GTEKW C_EXPRESSION prec GTEKW)
    (C_EXPRESSION$1.t=
      CGreatEqualExp(
        C_EXPRESSION$2.t,
        C_EXPRESSION$3.t
            );)

  |    (C_EXPRESSION LTEKW C_EXPRESSION prec LTEKW)
    (C_EXPRESSION$1.t=
      CLessEqualExp(
        C_EXPRESSION$2.t,
        C_EXPRESSION$3.t
          );)

  |    (C_EXPRESSION NEQVKW C_EXPRESSION prec NEQVKW)
    (C_EXPRESSION$1.t=
      CNotEquivExp(
        C_EXPRESSION$2.t,
        C_EXPRESSION$3.t
          );)

  |    (C_EXPRESSION  ANDKW  C_EXPRESSION prec ANDKW)
    (C_EXPRESSION$1.t=
      CAndExp(
        C_EXPRESSION$2.t,
        C_EXPRESSION$3.t
            );)

  |    (C_EXPRESSION ORKW C_EXPRESSION prec ORKW)
    (C_EXPRESSION$1.t=
      COrExp(
        C_EXPRESSION$2.t,
        C_EXPRESSION$3.t
          );)

  |    (C_EXPRESSION XORKW C_EXPRESSION prec XORKW)
    (C_EXPRESSION$1.t=
      CXorExp(
        C_EXPRESSION$2.t,
        C_EXPRESSION$3.t);)

  /* ARITHMETIC_EXPRESSION */

  |    (INTEGER)
    (C_EXPRESSION.t=CInteger(INTEGER.t);)

  |    (INTEGER '.' INTEGER)
    (C_EXPRESSION.t=CReal(
            INTEGER$1.t,
            INTEGER$2.t
                 );)


  |    (C_EXPRESSION '+' C_EXPRESSION prec '+')
    (C_EXPRESSION$1.t=
      CPlusExp(
        C_EXPRESSION$2.t,
        C_EXPRESSION$3.t
            );)

  |    (C_EXPRESSION '-' C_EXPRESSION prec '-')
    (C_EXPRESSION$1.t=
      CMinusExp(
        C_EXPRESSION$2.t,
```

```
      C_EXPRESSION$3.t
      );}

|   (C_EXPRESSION '*' C_EXPRESSION prec '*')
  {C_EXPRESSION$1.t=
    CTimesExp(
      C_EXPRESSION$2.t,
      C_EXPRESSION$3.t
      );}

|   (C_EXPRESSION '/' C_EXPRESSION prec '/')
  {C_EXPRESSION$1.t=
    CDivExp(
      C_EXPRESSION$2.t,
      C_EXPRESSION$3.t
          );}

|   ('+' C_EXPRESSION prec '+')
  {C_EXPRESSION$1.t=
    CPositiveExp(
      C_EXPRESSION$2.t
            );}

|   ('-' C_EXPRESSION prec '-')
  {C_EXPRESSION$1.t=
    CNegativeExp(
      C_EXPRESSION$2.t
            );}

|   (ABSKW '('C_EXPRESSION')' prec ABSKW)
  {C_EXPRESSION$1.t=
    CAbsExp(
      C_EXPRESSION$2.t
          );}

|   (C_EXPRESSION MODKW C_EXPRESSION prec MODKW)
  {C_EXPRESSION$1.t=
    CModExp(
      C_EXPRESSION$2.t,
      C_EXPRESSION$3.t
          );}

|   (C_EXPRESSION REMKW C_EXPRESSION prec REMKW)
  {C_EXPRESSION$1.t=
    CRemExp(
      C_EXPRESSION$2.t,
      C_EXPRESSION$3.t
          );}

|   (C_EXPRESSION EXPKW C_EXPRESSION prec EXPKW)
  {C_EXPRESSION$1.t=
    CExponentExp(
      C_EXPRESSION$2.t,
      C_EXPRESSION$3.t
         );}

/* STRING EXPRESSION */

|   (C_EXPRESSION '&' C_EXPRESSION prec '&')
  {C_EXPRESSION$1.t=
    CConcatExp(
      C_EXPRESSION$2.t,
      C_EXPRESSION$3.t
          );}

;
```

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center 8725 John J. Kingman Road, Suite 0944 Fort Belvoir, VA 22060	2
2.	Dudley Knox Library Naval Postgraduate School 411 Dyer Road Monterey, CA 93943	2
3.	Center for Naval Analysis 4401 Ford Avenue Alexandria, VA 22302	1
4.	Dr. Ted Lewis, Chairman, Code CS/L Computer Science Department Naval Postgraduate School Monterey, CA 93943	1
5.	Chief of Naval Research 800 North Quincy Street Arlington, VA 22217	1
6.	Dr. Luqi, Code CS/Lq Computer Science Department Naval Postgraduate School Monterey, CA 93943	18
7.	Capt Scott Grosenheider 125 6th AVE S.W. LeMars, IA 51031	3
8.	The Grosenheider's 214 1st Street S.W. LeMars, IA 51031	1
9.	The Betsworth's 125 6th AVE S.W. LeMars, IA 51031	1
10.	Ada Joint Program Office OUSDRE (R&AT) The Pentagon Washington, DC 20301	1